

Plugin Directory

source: [database-toolset](#) / [trunk](#) / [admin](#) / [class-database-toolset-backup.php](#)

Last change on this file was 2964538, checked in by neoslab, 20 months ago
Updated to new version
File size: 11.4 KB

Line	
1	<?php
2	/**
3	* Class Database Toolset Backup
4	* The backup-specific functionality of the plugin
5	*
6	* @author NeosLab <contact@neoslab.com>
7	* @link https://neoslab.com
8	* @version 1.4.8
9	* @package Database_Toolset
10	*/
11	class Database_Toolset_Backup
12	{
13	/**
14	* Host where the database is located
15	*/
16	var \$host;
17	
18	/**
19	* Username used to connect to database
20	*/
21	var \$username;
22	
23	/**
24	* Password used to connect to database
25	*/
26	var \$passwd;
27	
28	/**
29	* Database to backup
30	*/
31	var \$dbName;
32	
33	/**
34	* Database charset
35	*/
36	var \$charset;
37	
38	/**
39	* Database connection
40	*/
41	var \$conn;
42	
43	/**
44	* Backup directory where backup files are stored
45	*/
46	var \$backupDir;
47	
48	/**
49	* Output backup file

Line	
50	*/
51	var \$backupFile;
52	
53	/**
54	* Use gzip compression on backup file
55	*/
56	var \$gzipBackupFile;
57	
58	/**
59	* Content of standard output
60	*/
61	var \$output;
62	
63	/**
64	* Disable foreign key checks
65	*/
66	var \$disableForeignKeyChecks;
67	
68	/**
69	* Batch size, number of rows to process per iteration
70	*/
71	var \$batchSize;
72	
73	/**
74	* Constructor initializes database
75	*/
76	public function __construct(\$host, \$username, \$passwd, \$dbName, \$charset = 'utf8')
77	{
78	\$rand = rand(11111111, 99999999);
79	
80	\$this->host = \$host;
81	\$this->username = \$username;
82	\$this->passwd = \$passwd;
83	\$this->dbName = \$dbName;
84	\$this->charset = \$charset;
85	\$this->conn = \$this->initializeDatabase();
86	\$this->backupDir = DBT_BACKUP_PATH ? DBT_BACKUP_PATH : '.';
87	\$this->backupFile = 'backup-' . \$this->dbName . '-' . date("YmdHis", time()) . '-' . \$rand . '.sql';
88	\$this->gzipBackupFile = defined('DBT_BACKUP_GZIP') ? DBT_BACKUP_GZIP : true ;
89	\$this->disableForeignKeyChecks = defined('DBT_BACKUP_KEY_CHECKS') ? DBT_BACKUP_KEY_CHECKS : true ;
90	\$this->batchSize = defined('DBT_BACKUP_BATCH_SIZE') ? DBT_BACKUP_BATCH_SIZE : 1000;
91	\$this->output = '';
92	}
93	
94	/**
95	* Initialize Database
96	*/
97	protected function initializeDatabase()
98	{
99	try
100	{
101	\$conn = mysqli_connect(\$this->host, \$this->username, \$this->passwd, \$this->dbName);
102	if(mysqli_connect_errno())
103	{
104	throw new Exception('ERROR connecting database: ' . mysqli_connect_error());
105	die ();
106	}
107	
108	if(!mysqli_set_charset(\$conn, \$this->charset))
109	{
110	mysqli_query(\$conn, 'SET NAMES ' . \$this->charset);
111	}
112	}

```

Line
113     catch (Exception $e)
114     {
115         print_r($e->getMessage());
116         die();
117     }
118
119     return $conn;
120 }
121
122 /**
123  * Backup the whole database or just some tables
124  * Use '*' for whole database or 'table1 table2 table3 ...'
125  * @param string $tables
126  */
127 public function backupTables($tables = '*')
128 {
129     try
130     {
131         if($tables == '*')
132         {
133             $tables = array();
134             $result = mysqli_query($this->conn, 'SHOW TABLES');
135
136             while($row = mysqli_fetch_row($result))
137             {
138                 $tables[] = $row[0];
139             }
140         }
141         else
142         {
143             $tables = is_array($tables) ? $tables : explode(' ', str_replace(' ', '', $tables));
144         }
145
146         $sql = "CREATE DATABASE IF NOT EXISTS `".$this->dbName."`; \n\n";
147         $sql.= "USE `".$this->dbName."`; \n\n";
148
149         if($this->disableForeignKeyChecks === true)
150         {
151             $sql.= "SET foreign_key_checks = 0; \n\n";
152         }
153
154         /**
155          * Iterate tables
156          */
157         foreach($tables as $table)
158         {
159             $this->obfPrint("Backing up `".$table."` table...".str_repeat('.', 50-strlen($table)), 0, 0);
160
161             $sql.= 'DROP TABLE IF EXISTS `'.$table.'`;';
162             $row = mysqli_fetch_row(mysqli_query($this->conn, 'SHOW CREATE TABLE `'.$table.'`'));
163             $sql.= "\n\n".$row[1]."; \n\n";
164
165             $row = mysqli_fetch_row(mysqli_query($this->conn, 'SELECT COUNT(*) FROM `'.$table.'`'));
166             $numRows = $row[0];
167
168             $numBatches = intval($numRows / $this->batchSize) + 1;
169
170             for($b = 1; $b <= $numBatches; $b++)
171             {
172                 $query = 'SELECT * FROM `'.$table.'` LIMIT '.$b * $this->batchSize - $this->batchSize).','.$this->batchSize;
173                 $result = mysqli_query($this->conn, $query);
174                 $realBatchSize = mysqli_num_rows ($result);

```

```

Line
175     $numFields = mysqli_num_fields($result);
176
177     if($realBatchSize !== 0)
178     {
179         $sql.= 'INSERT INTO `'.$table.` VALUES ';
180
181         for($i = 0; $i < $numFields; $i++)
182         {
183             $rowCount = 1;
184             while($row = mysqli_fetch_row($result))
185             {
186                 $sql.='(';
187                 for($j=0; $j<$numFields; $j++)
188                 {
189                     if(isset($row[$j]))
190                     {
191                         $row[$j] = addslashes($row[$j]);
192                         $row[$j] = str_replace("\n", "\\n", $row[$j]);
193                         $row[$j] = str_replace("\r", "\\r", $row[$j]);
194                         $row[$j] = str_replace("\f", "\\f", $row[$j]);
195                         $row[$j] = str_replace("\t", "\\t", $row[$j]);
196                         $row[$j] = str_replace("\v", "\\v", $row[$j]);
197                         $row[$j] = str_replace("\a", "\\a", $row[$j]);
198                         $row[$j] = str_replace("\b", "\\b", $row[$j]);
199
200                         if($row[$j] == 'true' or $row[$j] == 'false' or preg_match('/^-?[0-9]+$/', $row[$j]) or $row[$j] == 'NULL' or $row[$j] == 'null')
201                         {
202                             $sql.= $row[$j];
203                         }
204                         else
205                         {
206                             $sql.= ' '.$row[$j]. ' ';
207                         }
208                     }
209                     else
210                     {
211                         $sql.= 'NULL';
212                     }
213
214                     if($j < ($numFields-1))
215                     {
216                         $sql.= ',';
217                     }
218                 }
219
220                 if($rowCount == $realBatchSize)
221                 {
222                     $rowCount = 0;
223                     $sql.= ");\n";
224                 }
225                 else
226                 {
227                     $sql.= "),\n";
228                 }
229
230                 $rowCount++;
231             }
232         }
233
234         $this->saveFile($sql);
235         $sql = '';
236     }

```

```

Line
237         }
238
239         $sql.="\\n\\n";
240         $this->obfPrint('OK');
241     }
242
243     if($this->disableForeignKeyChecks === true)
244     {
245         $sql.= "SET foreign_key_checks = 1;\\n";
246     }
247
248     $this->saveFile($sql);
249
250     if($this->gzipBackupFile)
251     {
252         $this->gzipBackupFile();
253     }
254     else
255     {
256         $this->obfPrint('Backup file succesfully saved to '.$this->backupDir.'/'.$this->backupFile, 1, 1);
257     }
258 }
259 catch (Exception $e)
260 {
261     print_r($e->getMessage());
262     return false;
263 }
264
265 return $this->backupFile;
266 }
267
268 /**
269  * Save SQL to file
270  * @param string $sql
271  */
272 protected function saveFile(&$sql)
273 {
274     if(!$sql)
275     {
276         return false;
277     }
278
279     try
280     {
281         if(!file_exists($this->backupDir))
282         {
283             mkdir($this->backupDir, 0777, true);
284         }
285
286         file_put_contents($this->backupDir.'/'.$this->backupFile, $sql, FILE_APPEND | LOCK_EX);
287     }
288
289     catch (Exception $e)
290     {
291         print_r($e->getMessage());
292         return false;
293     }
294
295     return true;
296 }
297
298

```

```

Line
299 /**
300  * Gzip backup file
301  * @param integer $level GZIP compression level (default: 9)
302  * @return string New filename (with .gz appended) if success, or false if operation fails
303  */
304 protected function gzipBackupFile($level = 9)
305 {
306     if(!$this->gzipBackupFile)
307     {
308         return true;
309     }
310
311     $source = $this->backupDir.'/'.$this->backupFile;
312     $dest = $source.'.gz';
313
314     $this->obfPrint('Gzipping backup file to '.$dest.'... ', 1, 0);
315
316     $mode = 'wb'.$level;
317
318     if($fpOut = gzopen($dest, $mode))
319     {
320         if($fpIn = fopen($source, 'rb'))
321         {
322             while(!feof($fpIn))
323             {
324                 gzwrite($fpOut, fread($fpIn, 1024 * 256));
325             }
326
327             fclose($fpIn);
328         }
329         else
330         {
331             return false;
332         }
333
334         gzclose($fpOut);
335
336         if(!unlink($source))
337         {
338             return false;
339         }
340     }
341     else
342     {
343         return false;
344     }
345
346     $this->obfPrint('OK');
347     return $dest;
348 }
349
350 /**
351  * Prints message forcing output buffer flush
352  */
353 public function obfPrint($msg = '', $lineBreaksBefore = 0, $lineBreaksAfter = 1)
354 {
355     if(!$msg)
356     {
357         return false;
358     }
359
360     if($msg != 'OK' and $msg != 'KO')
361     {

```

Line	
362	<code>\$msg = date("Y-m-d H:i:s").' - ' . \$msg;</code>
363	<code>}</code>
364	
365	<code>\$output = '';</code>
366	
367	<code>if (php_sapi_name() != "cli")</code>
368	<code>{</code>
369	<code> \$lineBreak = "
";</code>
370	<code>}</code>
371	<code>else</code>
372	<code>{</code>
373	<code> \$lineBreak = "\n";</code>
374	<code>}</code>
375	
376	<code>if (\$lineBreaksBefore > 0)</code>
377	<code>{</code>
378	<code> for (\$i = 1; \$i <= \$lineBreaksBefore; \$i++)</code>
379	<code> {</code>
380	<code> \$output.= \$lineBreak;</code>
381	<code> }</code>
382	<code>}</code>
383	
384	<code>\$output.= \$msg;</code>
385	
386	<code>if (\$lineBreaksAfter > 0)</code>
387	<code>{</code>
388	<code> for (\$i = 1; \$i <= \$lineBreaksAfter; \$i++)</code>
389	<code> {</code>
390	<code> \$output.= \$lineBreak;</code>
391	<code> }</code>
392	<code>}</code>
393	
394	<code>\$this->output.= str_replace('
', '\n', \$output);</code>
395	<code>echo \$output;</code>
396	
397	<code>if (php_sapi_name() != "cli")</code>
398	<code>{</code>
399	<code> if (ob_get_level() > 0)</code>
400	<code> {</code>
401	<code> ob_flush();</code>
402	<code> }</code>
403	<code>}</code>
404	
405	<code>\$this->output .= " ";</code>
406	<code>flush();</code>
407	<code>}</code>
408	
409	<code>/**</code>
410	<code> * Returns full execution output</code>
411	<code>*/</code>
412	<code>public function getOutput()</code>
413	<code>{</code>
414	<code> return \$this->output;</code>
415	<code>}</code>
416	<code>}</code>
417	
418	<code>?></code>

[About](#)

[News](#)

[Hosting](#)

[Donate](#)

[Swag](#)

[Documentation](#)

[Developers](#)

[Get Involved](#)

[Learn](#)

[Showcase](#)

[Plugins](#)

[Themes](#)

[Patterns](#)

[WordCamp](#)

[WordPress.TV](#)

[BuddyPress](#)

[bbPress](#)

[WordPress.com](#)

[Matt](#)

[Privacy](#)

[Public Code](#)

