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

```
<?php
/**
 * Class Database Toolset Backup
 * The backup-specific functionality of the plugin
 *
 * @author NeosLab <contact@neoslab.com>
 * @link https://neoslab.com
 * @version 1.4.8
 * @package Database_Toolset
*/
class Database_Toolset_Backup
{
    /**
     * Host where the database is located
    */
    var $host;

    /**
     * Username used to connect to database
    */
    var $username;

    /**
     * Password used to connect to database
    */
    var $passwd;

    /**
     * Database to backup
    */
    var $dbName;

    /**
     * Database charset
    */
    var $charset;

    /**
     * Database connection
    */
    var $conn;

    /**
     * Backup directory where backup files are stored
    */
    var $backupDir;

    /**
     * Output backup file
```

```
    */
    var $backupFile;

    /**
     * Use gzip compression on backup file
    */
    var $gzipBackupFile;

    /**
     * Content of standard output
    */
    var $output;

    /**
     * Disable foreign key checks
    */
    var $disableForeignKeyChecks;

    /**
     * Batch size, number of rows to process per iteration
    */
    var $batchSize;

    /**
     * Constructor initializes database
    */
    public function __construct($host, $username, $passwd, $dbName, $charset = 'utf8')
    {
        $rand = rand(11111111, 99999999);

        $this->host = $host;
        $this->username = $username;
        $this->passwd = $passwd;
        $this->dbName = $dbName;
        $this->charset = $charset;
        $this->conn = $this->initializeDatabase();
        $this->backupDir = DBT_BACKUP_PATH ? DBT_BACKUP_PATH : '.';
        $this->backupFile = 'backup-'.$this->dbName.'-'.date("YmdHis", time()).'-'.$rand.'.sql';
        $this->gzipBackupFile = defined('DBT_BACKUP_GZIP') ? DBT_BACKUP_GZIP : true;
        $this->disableForeignKeyChecks = defined('DBT_BACKUP_KEY_CHECKS') ? DBT_BACKUP_KEY_CHECKS : true;
        $this->batchSize = defined('DBT_BACKUP_BATCH_SIZE') ? DBT_BACKUP_BATCH_SIZE : 1000;
        $this->output = '';
    }

    /**
     * Initialize Database
    */
    protected function initializeDatabase()
    {
        try
        {
            $conn = mysqli_connect($this->host, $this->username, $this->passwd, $this->dbName);
            if(mysqli_connect_errno())
            {
                throw new Exception('ERROR connecting database: '.mysqli_connect_error());
                die();
            }

            if(!mysqli_set_charset($conn, $this->charset))
            {
                mysqli_query($conn, 'SET NAMES '.$this->charset);
            }
        }
```

```
        catch (Exception $e)
        {
            print_r($e->getMessage());
            die();
        }

        return $conn;
    }

    /**
     * Backup the whole database or just some tables
     * Use '*' for whole database or 'table1 table2 table3 ...'
     * @param string $tables
    */
    public function backupTables($tables = '*')
    {
        try
        {
            if($tables == '*')
            {
                $tables = array();
                $result = mysqli_query($this->conn, 'SHOW TABLES');

                while($row = mysqli_fetch_row($result))
                {
                    $tables[] = $row[0];
                }
            }
            else
            {
                $tables = is_array($tables) ? $tables : explode(',', str_replace(' ', '', $tables));
            }

            $sql = "CREATE DATABASE IF NOT EXISTS `".$this->dbName."`;\n\n";
            $sql.= "USE `".$this->dbName."`;\n\n";

            if($this->disableForeignKeyChecks === true)
            {
                $sql.= "SET foreign_key_checks = 0;\n\n";
            }

            /**
             * Iterate tables
            */
            foreach($tables as $table)
            {
                $this->obfPrint("Backing up `".$table."` table...".str_repeat('.', 50-strlen($table)), 0, 0);

                $sql.= 'DROP TABLE IF EXISTS `'.$table.'`;';
                $row = mysqli_fetch_row(mysqli_query($this->conn, 'SHOW CREATE TABLE `'.$table.'`'));
                $sql.= "\n\n".$row[1].";\n\n";

                $row = mysqli_fetch_row(mysqli_query($this->conn, 'SELECT COUNT(*) FROM `'.$table.'`'));
                $numRows = $row[0];

                $numBatches = intval($numRows / $this->batchSize) + 1;

                for($b = 1; $b <= $numBatches; $b++)
                {
                    $query = 'SELECT * FROM `'.$table.'` LIMIT '.($b * $this->batchSize - $this->batchSize).','.$this->batchSize;
                    $result = mysqli_query($this->conn, $query);
                    $realBatchSize = mysqli_num_rows ($result);
```

```
                    $numFields = mysqli_num_fields($result);

                    if($realBatchSize !== 0)
                    {
                        $sql.= 'INSERT INTO `'.$table.'` VALUES ';

                        for($i = 0; $i < $numFields; $i++)
                        {
                            $rowCount = 1;
                            while($row = mysqli_fetch_row($result))
                            {
                                $sql.='(';
                                for($j=0; $j<$numFields; $j++)
                                {
                                    if(isset($row[$j]))
                                    {
                                        $row[$j] = addslashes($row[$j]);
                                        $row[$j] = str_replace("\n","\\n",$row[$j]);
                                        $row[$j] = str_replace("\r","\\r",$row[$j]);
                                        $row[$j] = str_replace("\f","\\f",$row[$j]);
                                        $row[$j] = str_replace("\t","\\t",$row[$j]);
                                        $row[$j] = str_replace("\v","\\v",$row[$j]);
                                        $row[$j] = str_replace("\a","\\a",$row[$j]);
                                        $row[$j] = str_replace("\b","\\b",$row[$j]);

                                        if($row[$j] == 'true' or $row[$j] == 'false' or preg_match('/^-?[0-9]+$/', $row[$j]) or $row[$j] == 'NULL' or $row[$j] == 'null')
                                        {
                                            $sql.= $row[$j];
                                        }
                                        else
                                        {
                                            $sql.= '"'.$row[$j].'"' ;
                                        }
                                    }
                                    else
                                    {
                                        $sql.= 'NULL';
                                    }

                                    if($j < ($numFields-1))
                                    {
                                        $sql.= ',';
                                    }
                                }

                                if($rowCount == $realBatchSize)
                                {
                                    $rowCount = 0;
                                    $sql.= ");\n";
                                }
                                else
                                {
                                    $sql.= "),\n";
                                }

                                $rowCount++;
                            }
                        }

                        $this->saveFile($sql);
                        $sql = '';
                    }
```

```
                }

                $sql.="\n\n";
                $this->obfPrint('OK');
            }

            if($this->disableForeignKeyChecks === true)
            {
                $sql.= "SET foreign_key_checks = 1;\n";
            }

            $this->saveFile($sql);

            if($this->gzipBackupFile)
            {
                $this->gzipBackupFile();
            }
            else
            {
                $this->obfPrint('Backup file succesfully saved to '.$this->backupDir.'/'.$this->backupFile, 1, 1);
            }
        }
        catch (Exception $e)
        {
            print_r($e->getMessage());
            return false;
        }

        return $this->backupFile;
    }

    /**
     * Save SQL to file
     * @param string $sql
    */
    protected function saveFile(&$sql)
    {
        if(!$sql)
        {
            return false;
        }

        try
        {
            if(!file_exists($this->backupDir))
            {
                mkdir($this->backupDir, 0777, true);
            }

            file_put_contents($this->backupDir.'/'.$this->backupFile, $sql, FILE_APPEND | LOCK_EX);

        }

        catch (Exception $e)
        {
            print_r($e->getMessage());
            return false;
        }

        return true;
    }

```

```
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         {
```

```
            $msg = date("Y-m-d H:i:s").' - '.$msg;
        }

        $output = '';

        if(php_sapi_name() != "cli")
        {
            $lineBreak = "<br />";
        }
        else
        {
            $lineBreak = "\n";
        }

        if($lineBreaksBefore > 0)
        {
            for($i = 1; $i <= $lineBreaksBefore; $i++)
            {
                $output.= $lineBreak;
            }
        }

        $output.= $msg;

        if($lineBreaksAfter > 0)
        {
            for($i = 1; $i <= $lineBreaksAfter; $i++)
            {
                $output.= $lineBreak;
            }
        }

        $this->output.= str_replace('<br />', '\n', $output);
        echo $output;

        if(php_sapi_name() != "cli")
        {
            if(ob_get_level() > 0)
            {
                ob_flush();
            }
        }

        $this->output .= " ";
        flush();
    }

    /**
     * Returns full execution output
    */
    public function getOutput()
    {
        return $this->output;
    }
}

?>
```

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