```php
<?php
/**
 * Database class which connects to database and processes data with debug and error checking
 * * This class extends the MySQLi class
 *
 * @author  Floris van Enter
 * @link  http://floris.vanenter.nl
 * @DocLink     http://bit.ly/mpjbYi
 * @email  floris@entermi.nl
 * @license  Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
 *
 * @file  _class.mysqli.php
 * @location inc/classes
 * @version  1.0
 * @date  2011-06-02
 *
 * @depending PHP / MySQLi / class MySQLi
 *
 * Copyright (c) 2011
 */


//  Create the class by extending the parent
class extMysqli extends Mysqli
{
    /**
     * properties of the class
     *
     * @properties          input       variables
     * $this->debug          private     enables the extensive log function
     * $this->prefix         private     table prefix, unique tablenames  for multiple instances
     * $errorOccured         private     saves TRUE/FALSE to check if an error occured in the class
     * $this->errorMsg       private     saves the user error message
     * $this->errorMsgSql    private     saves the SQL error message
     *
     */


    private $debug = FALSE;
    private $prefix;
    private $errorOccured = FALSE;
    private $errorMessage;
    private $errorMessageSql;

    public function __construct($dbhost, $dbuser, $dbpass, $dbname, $dbprefix, $dbtable)
    {
        /**
         * method which is called by initiation
         * function is to build a connection, test DB & tables and handle errors when they occur
         *
         * @arguments input variables
         * dbhost   -  MySQL hostname
         * dbuser   -  MySQL user
         * dbpass   -  MySQL user password
         * dbname   -  MySQL database
         * dbprefix -  Table prefix
         * dbtable  -  Table to test for
         *
         * @returns  output variables
         * nothing
         */


        //  run the construct method of the parent: MySQLi
        @parent::__construct($dbhost, $dbuser, $dbpass);

        //  check on connection errors and handle it
        if($this->connect_error) {
            $this->setError('Connection to the MySQL db has failed.',
                            $this->connect_error,
                            NULL);
            return FALSE;
        }

        //  check on existence database and handle it
        if(!@$this->select_db($dbname)) {
            @$this->setError('Selecting the database ' . $dbname . ' has failed.',
                                                        $this->error, 'USE ' . $dbname);
            return FALSE;
        }

        //  check on existence tables and handle it
        $query = "SELECT 56 FROM `" . $dbprefix . $dbtable . "`";
        if(!@parent::query($query)) {
            @$this->setError('Selecting the table ' . $dbprefix . $dbtable . ' has failed.',
                                                                      $this->error, $query);
            return FALSE;
        }

        //  save the table prefix in the object
        $this->prefix = $dbprefix;
    }  #  end constructor


    public function __destruct()
    {
        /**
         * method which is called by object deletion
         * function is to cleanup the connection and free resources
         *
         * @arguments input variables
         * nothing
         *
         * @returns  output variables
         * nothing
         */


        @$this->close();
    }  #  end destructor


    protected function getTime()
    {
        /**
         * method which is called at request in the class
         * function is to get the seconds + microseconds sinds Unix epoch
         *
         * @arguments input variables
         * nothing
         *
         * @returns  output variables
         * seconds + microseconds
         */


        list($time['micro'], $time['sec']) = explode(" ", microtime());
        return (float)$time['micro'] + (float)$time['sec'];
    }


    public function setEvent($scope, $type, $method, $error=NULL, $query=NULL)
    {
        /**
         * method which is called at request in the class or in the application
         * function saves certain events in the table
         *
         * @arguments input variables
         * scope    - where did the event occur (class/function/file/etc)
         * type     - err = error, log = logged, chk = warning
         * method   - what triggered the event?
         * error    - what error message is created?
         * query    - what query was used to create the error
         *
         * @returns  output variables
         * nothing
         */


        //  check & fix the variables on characters that creates syntax errors
        $query = "INSERT INTO `" . $this->prefix . "eventLog` "
               . "       (`ID`, `dateTime`, `scope`, `type`, `method`, `error`, `query`) "
               . "VALUES "
               . "       (NULL, NOW(), '".$scope."', '".$type."', ".$method."', '"
               .                                                  $error."', '".$query."')";
        @parent::query($query);
    }  #  end setEvent


    public function setDebug($var)
    {
        /**
         * method which is called in/outside the class
         * function enables the debug option
         *
         * @arguments input variables
         * var         - variable to turn debugging on/off
         *
         * @returns  output variables
         * this->debug - debug status
         *
         */

        //  convert a string to lower case to ensure compatibility
        if(is_string($var))
        {   $var = strtolower($var); }

        //  depending on the value set $this->debug
        switch ($var) {
            case 'on':  $this->debug = TRUE;    break;
            case 1:     $this->debug = TRUE;    break;
            case TRUE:  $this->debug = TRUE;    break;
            case 'off': $this->debug = FALSE;   break;
            case 0:     $this->debug = FALSE;   break;
            case FALSE: $this->debug = FALSE;   break;
            default:    $this->debug = FALSE;   break;
        }

        //  returning the debug status
        return $this->debug;
    }  #  end debug


    protected function setError($message, $sqlError, $query=NULL)
    {
        /**
         * method runs at request after errors
         * function saves errors in properties occured during database actions
         *
         * @arguments input variables
         * message      -  Message translated specific for the user
         * sqlMessage   -  SQL error message for extra information
         * query        -  Query where it went wrong
         *
         * @returns  output variables
         * nothing
         *
         */


        //  set the event in the database
        $this->setEvent('extMysqli', 'err', $message, $sqlError, $query);

        //  set the properties
        $this->errorOccured = TRUE;
        $this->errorMsg     = $message;
        $this->errorMsgSql  = $sqlMessage;
    }  #  end setError


    public function getError($type='sql')
    {
        /**
         * method runs at request to get the errors
         * function saves errors in properties occured during database actions
         *
         * @arguments input variables
         * type         -  Which error needs to returned? sql or user
         *
         * @returns  output variables
         * this->errorMsg or $this->errorMsgSql
         *
         */


        switch ($type) {
            case 'sql':     return $this->errorMsgSql;  break;
            case 'user':    return $this->errorMsg;     break;
            default:        return $this->errorMsgSql;  break;
        }
    }  #  end getError


    public function query($query)
    {
        /**
```

```
         * method runs at request from inside or outside the class
         * function uses the parent::query to query the database and handles errors
         *
         * @arguments input variables
         * query    -  Query to perform on a database
         *
         * @returns  output variables
         * result or FALSE - depending on the succes of the query
         *
         */


        //  Start the time to check on slow queries
        $time['start'] = $this->getTime();

        //  Clean up the query, remove newlines & trim query
        $query = str_replace("\n", " ", $query);
        $query = trim($query);

        //  if debug is enabled write the query in the eventlog
        if($this->debug) {
            $this->setEvent('extMysqli', 'log', 'debug-feature', NULL, $query); }

        if($result = @parent::query($query)) {
            //  be sure that no error is set
            unset($this->errorMsg);
            unset($this->errorMsgSql);
            $this->errorOccured = FALSE;

            //  check on slow queries
            $time['total'] = round($this->getTime() - $time['start'], 2);

            if($time['total'] > 4) {
                $this->setEvent('extMysqli', 'chk', "slow-query: " . $time['total'],
                                                                NULL, $query);
            }
            return $result;
        } else {
            //  set the error flags and log it
            @$this->setError('Error occurred executing SQL query', $this->error, $query);
            return FALSE;
        }
    }  #  end query


    public function queryArray($query)
    {
        /**
         * method runs at request from inside or outside the class
         * function uses the this->query to query and
         *              returns the all rows he gets in a 2 dimensional array
         *
         * @arguments input variables
         * query    -  Query to perform on a database
         *
         * @returns  output variables
         * 2 dimensional array or FALSE - depending on the succes of the query
         *
         */


        //  query the query
        if($result = $this->query($query)) {
            $i = 1;
            while($row = $result->fetch_assoc()) {
                foreach ($row as $key => $value) {
                    $array[$i][$key] = $value;
                }
            $i++;
            }

            return $array;
        }
        return FALSE;
    }  #  end queryArray


    public function queryRow($query)
    {
        /**
         * method runs at request from inside or outside the class
         * function uses the this->query to query and returns the first row he gets
         *
         * @arguments input variables
         * query    -  Query to perform on a database
         *
         * @returns  output variables
         * array or FALSE - depending on the succes of the query
         *
         */


        //  query the query
        if($result = $this->query($query)) {
            //  get the results in 1 array
            if($row = $result->fetch_assoc()) {
                return $row;
            }
        }
        return FALSE;
    }  #  end queryRow


    public function escapeString($var)
    {
        /**
         * method runs at request from inside/outside the class
         * function uses the this->real_escape_string to clean the variables
         *
         * @arguments input variables
         * var    -  Query to perform on a database
         *
         * @returns  output variables
         * result or FALSE - depending on the query
         *
         */


        //  check if it is an array
        if(is_array($var)) {
            foreach ($var as $key => $value) {
                if(is_string($var[$key])) {
                    $var[$key] = $this->real_escape_string($value); }
            }
        } else {
            if(is_string($var)) {
                $var = $this->real_escape_string($var); }
        }
        return $var;
    }    #  end sqlClean


    public function maintenance($var)
    {
        /**
         * method runs at request from inside/outside the class
         * function that performs table optimization and
         *
         * @arguments input variables
         * var    -  Query to perform on a database
         *
         * @returns  output variables
         * result or FALSE - depending on the query
         *
         */


        //  get the tables with the prefix in an array
        $query = "SHOW TABLES LIKE \'" . $this->prefix . "%\'";
        $result = $this->query($query);

        while($row = $result->fetch_row()) {
            //  create a table variable with the tablenames
            if(!isset($tables)) {
                $tables = $row[0]; }
            else {
                $tables .= ', ' . $row[0]; }
        }

        //  repair tables, repairs defected tables and does nothing with healthy tables
        $query = "ANALYZE TABLE  " . $tables;
        $this->query($query);

        //  analyze tables, analyzes and stores the key distribution for a table.
        $query = "REPAIR TABLE  " . $tables;
        $this->query($query);

        //  optimizes tables, reclaims unused space and defragments the data file
        $query = "OPTIMIZE TABLE  " . $tables;
        $this->query($query);
    }    #  end maintenance
}   #  end extMysqli class
```