PHP Class Documentation

mysqlez

Summary

The `mysqlez` class extends the `mysqli` class packaged with all PHP versions 5 and 7. The primary benefit to using `mysqlez` is the added `parameterized_query` method, which uses prepared SQL statements, parameterized input, and convenient, context-aware return types.

The `mysqlez` class also includes default database connection parameters to make initialization more convenient in all scripts utilizing `mysqlez`.

Connecting

Configure connection parameters in the `mysqlez.php` file:

```
private $dbHost = 'localhost';
private $dbUser = 'root';
private $dbPass = 'root';
private $dbName = 'example';
```

Once mysqlez is configured, connecting to the database is simple:

```
require_once('mysqlez.php');
$db = new mysqlez();
```

Parameterized Queries

Function Structure:

```
mixed mysqlez::parameterized_query( string $sql, [ array $params ] );
```

Example Usage:

```
$result = $db->parameterized_query('SELECT * FROM `user` WHERE `id`=?', 'foo');
$errors = $db->errors;
```

Return values vary depending on the [successful] query type.

- SELECT
  - An array of rows, each row is an associative array wherein the column names are used as the array keys.
- INSERT
  - For tables with an `AUTO_INCREMENT` id column, the new row's id is returned.
  - Otherwise it returns `true` when successful.
- UPDATE
  - The number of rows updated.
- DELETE
  - The number of rows deleted.
- All query failures return `false`, and the error details are saved in the `errors` array.

The `errors` array holds one entry for each error encountered (usually only one, but not necessarily so). Each error includes three properties:

```
{string} operation - The operation in which the error occurred.
{number} errno     - The mysqli error number.
{string} error     - The mysqli error description.
```

Examples

```
# Setup #

require_once('mysqlez.php');
$db = new mysqlez();

# SELECT #

$fruit = $db->parameterized_query(
  'SELECT * FROM `fruit` WHERE `color`=?',
  $color
);

echo "Found " . count( $fruit ) . " fruit with the color $color.\n";

# INSERT #

$my_fruit = array(
  'name'  => 'apple',
  'color' => 'red'
);

$id = $db->parameterized_query(
  'INSERT INTO `fruit` (`name`,`color`) VALUES (?,?)',
  $my_fruit
);

if ( $id ) {
  echo "Added your fruit with ID number $id.\n";
} else {
  echo "Could not add your fruit.";
  foreach ( $db->errors as $err ) {
    echo " " . $err['error'];
  }
  echo "\n";
}
```

```
# UPDATE #

$my_fruit = array(
  'name'  => 'apple',
  'color' => 'green'
);

$effect = $db->parameterized_query(
  'UPDATE `fruit` SET `color`=? WHERE `name`=?',
  array( $my_fruit['color'], $my_fruit['name'] )
);

if ( $effect ) {
  echo "Fruit updated.\n";
} else {
  echo "Could not update your fruit.\n";
}

# DELETE #

$dont_like = 'green';

$effect = $db->parameterized_query(
  'DELETE FROM `fruit` WHERE `color`=?',
  $dont_like
);

if ( $effect ) {
  echo "$dont_like fruit deleted!\n";
} else {
  if ( $effect === 0 ) {
    echo "No $dont_like fruit to delete.\n";
  } else {
    echo "Could not delete $dont_like fruit!\n";
    var_dump( $db->errors );
  }
}
```

Comparison with MySQLi

For comparison, here's the equivalent code using the PHP standard `mysqli`:

```
if (! $db = new mysqli( $dbHost, $dbUser, $dbPass, $dbName ) {
  # Handle connection errors here...
}

$sql    = 'SELECT * FROM `user` WHERE `id`=?';
$id     = 'foo';

# Prepare the query. #
if (! $stmt = $mysqli->prepare( $sql ) ) {
  # Error handling, for example:
  echo "Failed to prepare mysqli query: " . $mysqli->errno . ' ' . $this->error;
  exit;
}

# Bind the parameters. #
if (! $stmt->bind_param( 's', $id )) {
  # More error handling...
}

# Execute the query. #
if (! $stmt->execute()) {
  # More error handling...
}

# Identify the columns in the SELECT result set. #
$fields = array();
foreach ( $stmt->result_metadata()->fetch_fields() as $field) {
  $fields[] = $field->name;
}

# Bind the results variables. #
$row = array();
$bind_results = array();
foreach ($fields as $fieldName) {
  $row[$fieldName] = null;
  $bind_results[] = &$row[$fieldName];
}
if (! call_user_func_array( array($stmt, "bind_result"), $bind_results )) {
  # Even more error handling...
}

# Compile and return the results as an associative array. #
$data = array();
while ($stmt->fetch()) {
  array_push($data, $row);
}
return $data;
```

Note that the standard `mysqli` requires each result column to be bound to its own variable reference. This is very inconvenient should you want to select all the column in a table with the '`*`' character. The code above is written to adapt to any number of result columns. The same technique is included in the `mysqlez` extension.

If for some strange reason you didn't want to do any error checking, and always selected a known number of columns, you could get away with as little code as this:

```
$sql = 'SELECT `every`,`desired`,`column`,`name` FROM `user` WHERE `id`=?';
$id  = 'foo';

$stmt = $mysqli->prepare( $sql );
$stmt->bind_param( 's', $id );
$stmt->execute();
$stmt->bind_result( $every, $desired, $column, $name );
$data = array();
while ($stmt->fetch()) {
  $data[] = array(
    'every'   => $every,
    'desired' => $desired,
    'column'  => $column,
    'name'    => $name
  );
}
return $data;
```

But why give up flexibility and error checking when you could do everything included in the page-length example in four lines?

```
require_once('mysqlez.php');
$db = new mysqlez();
$result = $db->parameterized_query('SELECT * FROM `user` WHERE `id`=?', 'foo');
$errors = $db->errors;
```