

Getting Started

Instantiation

You can instantiate *ExtendedPdo* so that it uses lazy connection, or you can use *DecoratedPdo* to decorate an existing *PDO* instance.

Lazy Connection Instance

Instantiation is the same as with the native *PDO* class: pass a data source name, username, password, and driver options. There is one additional parameter that allows you to pass attributes to be set after the connection is made.

```
use Aura\Sql\ExtendedPdo;

$pdo = new ExtendedPdo(
    'mysql:host=localhost;dbname=test',
    'username',
    'password',
    [], // driver attributes/options as key-value pairs
    [] // queries to execute after connection
);
```

Whereas the native *PDO* connects on instantiation, *ExtendedPdo* does not connect immediately. Instead, it connects only when you call a method that actually needs the connection to the database; e.g., on `query()`.

If you want to force a connection, call the `connect()` method.

```
// does not connect to the database
$pdo = new ExtendedPdo(
    'mysql:host=localhost;dbname=test',
    'username',
    'password'
);

// automatically connects
$pdo->exec('SELECT * FROM test');

// explicitly forces a connection
$pdo->connect();
```

If you want to explicitly force a disconnect, call the `disconnect()` method.

```
// explicitly forces disconnection
$pdo->disconnect();
```

Doing so will close the connection by unsetting the internal *PDO* instance. However, calling an *ExtendedPdo* method that implicitly establishes a connection, such as `query()` or one of the `fetch*()` methods, will automatically re-connect to the database.

Decorator Instance

The *DecoratedPdo* class can be used to decorate an existing PDO connection with the *ExtendedPdo* methods. To do so, instantiate *DecoratedPdo* by passing an existing PDO connection:

```
use Aura\Sql\DecoratedPdo;

$pdo = new PDO(...);
$decoratedPdo = new DecoratedPdo($pdo);
```

The decorated *PDO* instance now provides all the *ExtendedPdo* functionality (aside from lazy connection, which is not possible since the *PDO* instance by definition has already connected).

Decoration of this kind can be useful when you have access to an existing *PDO* connection managed elsewhere in your application.

N.b.: The `disconnect()` method will **not work** on decorated *PDO* connections, since *DecoratedPdo* did not create the connection itself. You will need to manage the decorated *PDO* instance yourself in that case.

Array Quoting

The native *PDO* `quote()` method will not quote arrays. This makes it difficult to bind an array to something like an `IN (...)` condition in SQL. However, *ExtendedPdo* recognizes arrays and converts them into comma-separated quoted strings.

```
// the array to be quoted
$array = array('foo', 'bar', 'baz');

// the native PDO way:
// "Warning: PDO::quote() expects parameter 1 to be string, array given"
$pdo = new PDO(...);
$cond = 'IN (' . $pdo->quote($array) . ')';

// the ExtendedPdo way:
// "IN ('foo', 'bar', 'baz')"
$pdo = new ExtendedPdo(...);
$cond = 'IN (' . $pdo->quote($array) . ')';
```

The `perform()` Method

The `ExtendedPdo::perform()` method will prepare a query with bound values in a single step. Also, because the native *PDO* does not deal with bound array values, `perform()` modifies the query string to expand the array-bound placeholder into multiple placeholders.

```
// the array to be quoted
$array = array('foo', 'bar', 'baz');

// the statement to prepare
$stmt = 'SELECT * FROM test WHERE foo IN (:foo)';

// the native PDO way does not work (PHP Notice: Array to string conversion)
$pdo = new ExtendedPdo(...);
$stmt = $pdo->prepare($stmt);
$stmt->bindValue('foo', $array);

// the ExtendedPdo way allows a single call to prepare and execute the query.
// it quotes the array and expands the array placeholder directly in the
// query string.
$pdo = new ExtendedPdo(...);
$bind_values = array('foo' => $array);
$stmt = $pdo->perform($stmt, $bind_values);
echo $stmt->queryString;
// the query string has been modified by ExtendedPdo to become
// "SELECT * FROM test WHERE foo IN (:foo_1, :foo_2, :foo_3)"
```

Finally, note that array quoting works only via the `perform()` method, not on returned *PDOStatement* instances.