Tuza Docs Documentation

Versión 0.0.1

Yeshua Rodas

20 de septiembre de 2018

Índice general

Advertencia: Esta documentación se encuentra en proceso de construcción, todavía expuesta a cambios sustanciales y reestructuración de su contenido.

Tuza es una biblioteca PHP súper ligera[1] para el consumo de Base de Datos sobre PDO. Provee una API que abstrae (entre otros) las operaciones `CRUD` sobre bases de datos como métodos de una *instancia de conexión*.

Tuza es compatible con la mayoría de gestores de bases de datos soportados por PDO, cuatro de código abierto: **SQLite**, **MySQL**, **MariaDB**, **PostreSQL** y dos privativos: **SQL Server** y **Oracle**.

Tuza es una bifurcación de Medoo, pero no es compatible con este. La API de Tuza es ligeramente diferente pero algo más estática (para proveer de mayor consistencia) que la de Medoo.

Con Tuza es tremendamente sencillo consultar y persistir datos:

```php
<?php
use xibalba\tuza\DbConnection;

// Crear una instancia de conexión
$dbConnection = new DbConnection([
  'database_type' => 'sqlite',
  'database_file' => 'sqlitedb.db'
]);

// Obtener datos mediante SELECT
$data = $dbConnection->select('table', ['id', 'label'], ['LIMIT' => 9]);

// Crear nuevos datos...
$uniqid = uniqid();
$newData = [
  'id' => $uniqid,
  'label' => 'Etiqueta'
];

// ... y persistirlos
$dbConnection->insert('table', $newData);

// Actualizar datos
$dbConnection->update('table',
  ['label' => 'Etiqueta actualizada'],
  ['id' => $uniqid]
);

// Eliminar datos
$dbConnection->delete('table', ['id' => $uniqid]);
```

[1] Tuza se compone por sólo cinco archivos: Dos clases, dos interfaces, y un *trait*.

CAPÍTULO 1

Manual de usuario

1.1 Introducción

Existe un amplio abanico de gestores de bases de datos relacionales, desde la ligera SQLite hasta el potentísimo PostgreSQL. Naturalmente, muchas aplicaciones desarrolladas con PHP necesitan conectarse a estos gestores de bases de datos. El motor de PHP provee una interfaz ligera para acceder a bases de datos, la **PDO** (*PHP Data Objects* [Objetos de datos de PHP]). El soporte para PDO se provee mediante unos controladores específicos (*PDO Drivers*), que permiten a PHP interactuar con un gestor determinado. Dado que cada gestor tiene una características propias y específicas, PDO provee una capa de abstracción que expone una interfaz unificada independiente del gestor específico de conexión. Luego, es posible ejecutar sentencias SQL *relativamente* independiente del gestor que se utilice[1].

Si bien es cierto que gracias a PDO es posible preparar sentencias, enlazar datos con columnas y prevenir inyecciones de SQL, también es verdad que PDO es bastante genérica y las sentencias se ejecutan llamando al método `execute()`, no existiendo una abstracción nativa sobre las sentencias SQL más comunes (`INSERT`, `SELECT`, `UPDATE`, `DELETE`).

Cabe mencionar también que, buscando métodos para cumplir con patrones como el MVC, se han desarrollado potentes herramientas que proveen toda una capa orientada a objetos sobre las bases de datos: **los ORM**. Existen proyectos que indudablemente se benefician de la potencia de los ORM, pero otros no son tan grandes y utilizar un ORM resulta un tanto como matar moscas a cañonazos, o incluso, puede que se quiera desarrollar un ORM propio y no comenzar desde cero.

Tuza es precisamente esa capa sobre PDO que abstrae las operaciones `CRUD` exponiendo una interfaz orientada a objetos y una arquitectura ligera y extensible.

1.1.1 Arquitectura

Tuza se compone de cinco archivos que proveen una funcionalidad y esquema general para su utilización en cualquier proyecto. Tuza se provee listo para su uso, basta con crear una instancia de la clase `DbConnection` (con una configuración mínima) para realizar una conexión a un gestor soportado y comenzar a realizar operaciones. No obstante

[1] Como todo, tiene sus salvedades.

Tuza es más que una clase de conexión con una interfaz para las operaciones CRUD, es una biblioteca extensible con un esquema intuitivo y coherente.

La interfaz DbConnectionInterface

La abstracción mínima que debe tener una clase de conexión se establece en DbConnectionInterface. Dicha abstracción mínima no es más que la declaración de los métodos que abstraen las operaciones CRUD (insert(), select(), update(), delete()), un método de execución de sentencias (exec()), un método que devuelva la última consulra realizada (getLastQuery()) y la abstracción de algunas operaciones típicas sobre bases de datos (avg(), count(), max(), min(), sum()).

Tuza implementa dicha interfaz en la clase DbConnection, pero la idea de la interfaz es proveer al usuario de una base agnóstica para que, si por algún motivo prefiere realizar una implementación propia, no sólo parta de cero, sino que disponga de una base compatible con Tuza.

La interfaz DbConnctable

Esta interfaz define el comportamiento mínimo que debería tener cualquier clase que se pretenda que soporte conexiones a bases de datos. Provee la declaración de un sólo método: getDbConnection(). Y su definición debe devolver un objeto del tipo DbConnectionInterface.

El trait DbAware

Para la interfaz DbConnctable, se provee el trait DbAware que define una funcionalidad básica que cumple con dicha interfaz. El usuario bien puede optar por utilizar dicho trait, definir su propia implementación o una combinación de ambas.

La clase DbConnection

Esta es la clase que contiene la *magia*. Es precisamente una implementación de la interfaz DbConnectionInterface, con varias cosas más. El grueso de esta documentación es sobre la funcionalidad que provee esta clase.

La clase DbException

Finalmente, es inevitable que se desencadenen errores cuando se ejecutan sentencias. Por defecto, Tuza establece que el manejo de errores de PDO sea mediante Excepciones. DbConnection captura varias de las excepciones lanzadas por PDO y lanza excepciones DbException con información extra sobre la sentencia que falló.

1.2 Instalación y configuración

1.2.1 Instalación

– Por Hacer –

1.2.2 Configuración

Configurar Tuza es tremendamente sencillo, sólo se requiere proveer al constructor de `DbConnection` con un `array` de opciones. La configuración se define igual para todas las bases de datos soportadas, excepto para SQLite, que únicamente requiere definir el controlador y la ruta del archivo de la base de datos para funcionar.

Configurar SQLite

```
<?php
use xibalba\tuza\DbConnection;

// Crear una instancia de conexión
$dbConnection = new DbConnection([
  'database_type' => 'sqlite',
  'database_file' => 'sqlitedb.db'
]);
```

Configurar Tuza

La configuración mínima requiere que se defina:

- Controlador
- Servidor
- Nombre de la base de datos
- Credenciales de acceso

Controlador

Define uno de los controladores soportados por tuza, a continuación se muestra un listado con el nombre de cada base de datos soportada y la cadena de configuración correspondiente:

- SQLite: `sqlite`
- MariaDB: `mariadb`
- MySQL: `mysql`
- PostreSQL: `pgsql`
- Oracle: `oracle`
- Microsoft SQL Server: `sybase`, `dblib`, `sqlsrv`

Como puede verse, todas las cadenas son en minúsculas. La cadena debe definirse como se indica o de lo contrario Tuza no funcionará.

Las cadenas para MariaDB y MySQL pueden utilizarse indistintamente, es decir, se puede configurar `mysql` y uliziar MariDB.

Para SQL Server puede verse que se soportan tres cadenas distintas, esto es porque existen diversas formas de conectarse a SQL Server con PHP. El motor de PHP para Linux soporta `sybase` y `dblib`, no obstante Microsoft ha publicado su propio controlador tanto para Windows como la Linux, y en tal caso debe utilizarse `sqlsrv`.

1.3 Consultas y sentencias

Como se mencionó en la introducción, Tuza abstrae en la clase `DbConnection` las operaciones `CRUD` exponiendo una interfaz orientada a objetos de la misma mediante los siguientes métodos:

- `insert()`
- `update()`
- `delete()`
- `select()`

1.3.1 Insertar datos

`insert` (*string $table*, *array $datas*)

Abstrae la sentencia SQL del mismo nombre, y consecuentemente, el método `insert()` permite la inserción de datos en tablas.

Parámetros

- **`$table`** (*`string`*) – Cadena que especifíca en que tabla se insertarán los datos.
- **`$datas`** (*`array`*) – Arreglo de datos a insertar.

El Arreglo de datos es un conjunto «clave-valor» donde cada «clave» corresponde con el nombre de una columna de la tabla y cada «valor» con un dato a insertar.

```php
<?php

$data = [
    'id' => uniqid(),
    'titulo' => 'Algún título de ejemplo',
    'autor' => 'Nombre del Autor',
    'formato' => 'pdf',
    'paginas' => 100,
    'coste' => 200.5
];

$db->insert('tbl_tesis_ejemplo', $data);
```

También es posible insertar varios registros en una sola llamada al método `insert()`

```php
<?php

$datas = [
  [
    'id' => uniqid(),
    'titulo' => 'Algún título de ejemplo 1',
    'autor' => 'Nombre del Autor',
    'formato' => 'pdf',
    'paginas' => 100,
    'coste' => 200.5
  ],
  [
    'id' => uniqid(),
    'titulo' => 'Algún título de ejemplo 2',
    'autor' => 'Nombre del Autor',
```

(continues on next page)

(proviene de la página anterior)

```
    'formato' => 'epub',
    'paginas' => 100,
    'coste' => 200.5
  ]
];

$db->insert('tbl_tesis_ejemplo', $datas);
```

1.3.2 Actualizar datos

update (*string $table*, *array $datas*, *array $where*)

Abstrae la sentencia SQL del mismo nombre, y consecuentemente, el método `update()` permite la actualización de datos en tablas.

Parámetros

- **$table** (*string*) – Cadena que especifíca en que tabla se actualizarán los datos.
- **$data** (*array*) – Arreglo de datos a insertar.
- **$where** (*array*) – Arreglo de condiciones sobre la sentencia.

Al igual que con `insert()`, el Arreglo de datos es un conjunto «clave-valor» donde cada «clave» corresponde con el nombre de una columna de la tabla y cada «valor» con un dato a actualizar.

El arreglo de condiciones (`$where`) corresponde a la cláusa `WHERE`, y se estructura de manera similar al arreglo de datos, no obsante el arreglo de condiciones puedeser mucho más complejo. Para más detalles sobre como se estructura el arreglo de condiciones consulte la subsección de Condiciones correspondiente a `select()`.

```
<?php

$data = [
    'autor' => 'Nombre corregido del Autor',
    'formato' => 'pdf',
    'paginas' => 120,
    'coste' => 240
];

$condiciones = ['id' => 'xyz'];

$db->update('tbl_tesis_ejemplo', $data, $condiciones);
```

1.3.3 Eliminar datos

delete (*string $table*, *array $where*)

Abstrae la sentencia SQL del mismo nombre, y consecuentemente, el método `delete()` permite la eliminación de datos en tablas.

Parámetros

- **$table** (*string*) – Cadena que especifíca de que tabla se eliminarán datos.
- **$where** (*array*) – Arreglo de condiciones sobre la sentencia.

```php
<?php

$condiciones = ['id' => 'xyz'];
$db->delete('tbl_tesis_ejemplo', $condiciones);
```

1.3.4 Consultar datos

select (*string $table*, *$columns*, *$where*, *$join*)

Abstrae la sentencia SQL del mismo nombre y, consecuentemente, el método `select()` permite la consulta de datos en tablas.

Parámetros

- **$table** (*string*) – Cadena que especifíca de que tabla se consultarán datos.
- **$columns** (*string|array*) – Cadena o arreglo que especifíca las columnas a consultar.
- **$where** (*string|array*) – Cadena o arreglo de condiciones sobre la sentencia.
- **$join** (*array*) – Arreglo de uniones sobre la sentencia.

```php
<?php

$allData = $db->select('tbl_tesis_ejemplo', '*');
var_dump($allData);
```

El método `select()` es estructuralmente sencillo, pero muchas consultas requieren de condiciones *complejas*, no obstante gracias a que los parámetros de condiciones (`$where`) y unión (`$join`) son arreglos, permiten una gran potencia y flexibilidad en su estructuración.

Especificar columnas de consulta

El parámetro `$columns` soporta el asterisco (`'*'`) o `null` para indicar que se desea obtener **todas** las columnas de la tabla. Para especificar un conjunto de columnas a obtener se pasa un arreglo de cadenas con los nombres de las tablas:

```php
<?php

$allData = $db->select('tbl_tesis_ejemplo', ['id', 'titulo', 'autor']);
var_dump($allData);
```

Es posible también especificar un **alias** para cada columna adjuntando a la cadena, entre paréntesis, dicho *alias*. En el arreglo de retorno se utilizará como clave el *alias* y no el nombre de la columna:

```php
<?php

$allData = $db->select('tbl_tesis_ejemplo', ['id(código)', 'titulo(etiqueta)', 'autor
↪']);
var_dump($allData[0]['etiqueta']);
```

Especificar condiciones de consulta

Las condiciones se especifican mediante un arreglo. Tuza soporta la mayoría de condiciones de SQL.

```php
<?php

$columns = ['id', 'titulo', 'autor'];

$conditions = ['id' => 'xyz'];
// Equivalente a:
// WHERE id = 'xyz'

$allData = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
var_dump($allData[0]['etiqueta']);
```

Semejanzas (`LIKE`)

En SQL cuando se quiere especificar de una condición **no estricta**, sino más bien *semejante* se utiliza el operador `LIKE`. Este operador es abstraido en Tuza por el caracter de la virgulilla encerrado entre corchetes (`[~]`) y concatenado a la columna de comparación.

```php
<?php

$columns = ['id', 'titulo', 'autor'];

$conditions = ['autor[~]' => 'Ed'];
// Equivalente a:
// WHERE autor LIKE '%Ed%'

$byAuthoresLike = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
```

Para establecer la negación (`NOT LIKE`) se debe adjuntar el signo de admiración a la virgelilla.

```php
<?php

$columns = ['id', 'titulo', 'autor'];

$conditions = ['autor[!~]' => 'Ed'];
// Equivalente a:
// WHERE autor NOT LIKE '%Ed%'
```

Conjunciones (`AND`)

Las conjunciones se establecen con el operador `AND`, para establecer conjunciones en Tuza se declara un arreglo de elementos a conjuntar y se asignan al arreglo de condiciones en la clave `AND`.

```php
<?php

$columns = ['id', 'titulo', 'autor'];

$conjunctions = ['id' => 'xyz', 'coste[>=]' => 150, 'autor[~]' => 'abc'];
$conditions = ['AND' => $conjunctions];
// Equivalente a:
// WHERE id = 'xyz' AND coste >= 150 AND author LIKE '%abc%'

$fetchedData = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
var_dump($fetchedData);
```

Disyunciones (OR)

Las disyunciones se establecen con el operador `OR`, para establecer disyunciones en Tuza se declara un arreglo de elementos y se asignan al arreglo de condiciones en la clave `OR`.

```
<?php

$columns = ['id', 'titulo', 'autor'];

$conjunctions = ['id' => 'xyz', 'coste[>=]' => 150, 'autor[~]' => 'abc'];
$conditions = ['OR' => $conjunctions];
// Equivalente a:
// WHERE id = 'xyz' OR coste >= 150 OR author LIKE '%abc%'

$fetchedData = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
var_dump($fetchedData);
```

Disyunciones (IN)

Dado que también es posible establecer un conjunto de disyunciones mediante el operador `IN`, tuza soporta establecer el conjunto de elementos para dicho operador mediante un arreglo.

```
<?php

$columns = ['id', 'titulo', 'autor'];

$conjunctions = ['abc', 'xyz', 'vwm'];
$conditions = ['autor' => $conjunctions];
// Equivalente a:
// WHERE id IN('abc', 'xyz', 'vwm')

$fetchedData = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
var_dump($fetchedData);
```

Especificar uniones

Tuza soporta las cuantro uniones de SQL, para establecer una (o más) uniones se provee como cuarto argumento en el método `select()` un arreglo donde la clave especifica el tipo de unión y tabla, y el valor es un arreglo que relaciona los campos (columnas) de la tabla principal con la table de unión.

El tipo de unión se establece mediante la combinación de corchetes y signos angulares (mayor y menor que):

- `INNER JOIN`: `[><]`
- `FULL JOIN`: `[<>]`
- `LEFT JOIN`: `[>]`
- `RIGHT JOIN`: `[<]`

En el siguiente ejemplo supóngase que el campo `autor` no almacena el nombre sino el código de autor donde los datos de dicho autor (como el nombre, dirección, etc) se almacenan en otra tabla (relacionada por dicho campo) llamada `tbl_autores` y que se requiere desplegar el campo del nombre de los autores de los datos encontrados.

```php
<?php

$columns = ['tbl_tesis_ejemplo.id', 'tbl_tesis_ejemplo.titulo', 'tbl_autores.nombre'];

$conditions = ['author' => '6151'];
$join = ['[><]tbl_autores' => ['autor' => 'id']]

$fetchedData = $db->select('tbl_tesis_ejemplo', $columns, $conditions);
// Equivalente a:
// SELECT tbl_tesis_ejemplo.id, tbl_tesis_ejemplo.titulo, tbl_autores.nombre
// FROM tbl_tesis_ejemplo INNER JOIN tbl_autores ON tbl_autores.id = tbl_tesis_
↪ejemplo.autor
// WHERE tbl_tesis_ejemplo.autor = 6151;

var_dump($fetchedData);
```

Índice

Symbols