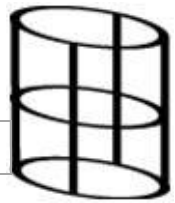
Manual de usuario de ObjectDB 4.0

Primeros pasos y la guía de APIs



Escrito por **Salvi Pascual**, traducido por **Yasmila Sealy**

Tabla de contenidos

Introducción

El acceso a bases de datos en PHP, poco compatible con el paradigma orientado a objetos, es complejo cuando se necesita almacenar y recuperar objetos en vez de variables por separado. Muchos programadores rompen con el encapsulamiento de lo que podría ser código bien modelado y fácilmente modificable; otros crean sus propios métodos de acceso a datos, lo cual cuesta gran parte del desarrollo del sistema en su totalidad.

ObjectDB crea una arquitectura general de acceso a datos, la cual (mediante la herencia) se puede incrementar para dar soporte a métodos específicos para cada aplicación. Con esta biblioteca el programador puede obtener, salvar, modificar (entre otras operaciones) objetos directamente desde la base de datos sin necesidad de memorizar la extensa API de PHP y sin poseer altos conocimientos del lenguaje SQL. ObjectDB ayuda a escribir un código más claro (y por ende mejor modificable), modelable bajo metodologías orientadas a objetos (actualmente las más utilizadas) y hace al programador completamente independiente del sistema de bases de datos a utilizar

ObjectDB está ideado para reducir el trabajo al mínimo posible. Al usarlo, operaciones agobiantes como crear relaciones entre tablas o buscar la última tupla insertada, se vuelven rutinarias y utilizan pocas líneas de código. Aunque pudiera pensarse que el trabajo de una capa extra de abstracción enlentece la aplicación, los mecanismos de caché de ObjectDB evitan operaciones redundantes y agilizan el trabajo, de manera que no se notan demoras de tiempo visibles.

Primeros pasos

A continuación se expone un ejemplo detallado de cómo configurar y poner en marcha una aplicación que realice simples accesos de lectura/escritura en una base de datos mediante ObjectDB. El mismo fue desarrollado con PHP v5.x y MySQL v5.x, deberá funcionar para otras versiones de MySQL y otros sistemas de bases de datos, dependiendo la biblioteca de ObjectDB que se utilice. Dado el pobre soporte que brinda PHP v4.x para aplicaciones orientadas a objetos, ObjectDB no funciona para estas versiones o inferiores.

Existen tres acciones básicas a realizar para crear un sencillo acceso a la base de datos:

1. Crear tablas en la base de datos.
2. Crear las clases de ObjectDB.
3. Configurar el acceso de ObjectDB a la base de datos.

Crear tablas en la base de datos

Al crear las tablas de la base de datos, el único requerimiento imprescindible es que la llave de la tabla sea numérica, autoincrementable y de nombre *id_nombretabla*. Puede crear una tabla soportada por ObjectDB, para una base de datos MySQL, con el siguiente script en lenguaje SQL:

```
CREATE TABLE student (
      id_student INT(11) KEY AUTO_INCREMENT,
      identification_number VARCHAR(11) NOT NULL,
      name varchar(50) NOT NULL,
      age INT(2) NULL,
      preferences TEXT
);
```

En el ejemplo anterior, note que el campo 'numero_identificacion' es una alternativa si se necesita un índice que no cumpla los requerimientos esperados por la biblioteca.

Crear las clases de ObjectDB

Todas las clases que almacenen datos deben extender de la clase *ODBObject*, definida en el núcleo de ObjectDB. Dicha clase, además de servir de base para encapsular la información, posee funcionalidades especiales que facilitan el trabajo del programador. En secciones más adelante se abundará el tema. Además de lo anterior, cada atributo de la clase se definirá como público y su nombre debe coincidir con el nombre de los campos de la tabla. Por convención también debe coincidir el orden de los atributos con el orden de los campos de la tabla, aunque lo anterior no es imprescindible.

A continuación una muestra en lenguaje PHP de como se definiría una clase para interactuar con la tabla definida anteriormente:

```
class student extends ODBObject{
      public $identification_number;
      public $name;
      public $age;
      public $preferences;
}
```

Aunque no sea imprescindible, por convención cada clase debe describirse en un archivo nominado *nombre_clase.php*; para el caso de la anterior, sería *student.php*. Este fichero puede comenzar con la inclusión del archivo *ODBObject.php* mediante la sentencia:

```
require_once "pathToFile/ODBObject.php";
```

Configurar el acceso de ObjectDB a la base de datos

Las primeras líneas de la clase *ODBConnection* son parámetros estáticos los cuales deben configurarse para cada servidor. Es posible configurar estos parámetros dinámicamente, aumentando la seguridad de las aplicaciones mediante la carga directa de un archivo.

```
...
/* start - configure */
static public $host   = "localhost";
static public $port   = "3306";
```

```
static public $dbname = "test";
static public $user   = "root";
static public $pass   = "";
/* end - configure */
...
```

La explicación a los parámetros anteriores se detalla a continuación:

```
$host
Dirección (DNS o IP) del servidor en el cual se encuentra la base de datos (localhost para la propia
máquina).
$port
Puerto para acceder a la base de datos.
$dbname
Nombre de la base de datos.
$user
Nombre de usuario para acceder a la base de datos.
$pass
Contraseña para acceder a la base de datos.
```

Una vez configurado la conexión con la base de datos, creada la tabla y la clase de ObjectDB, es posible realizar sencillos y funcionales accesos de lectura/escritura. Para ello es necesario crear un nuevo archivo, llamado *test.php* con el contenido siguiente:

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// crea y establece los valores para un nuevo estudiante
$student = new student();
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');

// guarda a este nuevo estudiante en la base de datos
$student->save();

// carga a todos los estudiantes guardados hasta el momento
$db = new objectDB();
$students_list = $db->getObjs('student');
```

```
// imprime la lista de estudiantes
echo 'List of students in database';
for($i=0; $i<count($students_list); $i++)
      echo $students_list[$i];
```

Lo anterior crea un nuevo estudiante, asigna valores al mismo, pasándolos como parámetros ordenados en base a los campos de la tabla (sin contar la llave) al método *setDataFromParamsList*, y luego salva el objeto en la base de datos. Por último crea un nuevo objeto *objectDB* y lo usa para cargar de la base de datos todos los estudiantes salvados hasta el momento y mostrarlos en pantalla. En secciones a continuación se explicará con mayor detalle los pasos de este proceso.

Estudios avanzados

Depurar el contenido de un objeto

ObjectDB permite mostrar el contenido de un objeto para simplificar a los programadores el proceso de depuración. Cualquier objeto que herede de *ODBObject*, al ser mostrado por pantalla (mediante un echo o printf) genera una tabla con la relación nombre de campo y valor. En el ejemplo a continuación se muestra el código entrado y la tabla resultante.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// crea y establece los valores para un nuevo estudiante
$student = new student();
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');

// imprime este objeto en modo de depuración
echo $student;
```

La tabla a continuación muestra como se vería la salida del script anterior

id_student	identification_number	name	age	preferences
1	3025	Salvi Pascual	24	color=orange,grossery=pie,serie=futurama

Es posible modificar la salida del algoritmo de depuración al sobrescribir el método *__toString* de la clase *ODBObject*. Luego sobrescribirlo, se mostrará al imprimir una variable la cadena de texto devuelta por el nuevo método. El siguiente código demuestra cómo podría hacerse esto para la clase student, definida en la sección anterior.

```
class student extends ODBObject{
	public $identification_number;
	public $name;
	public $age;
	public $preferences;

	function __toString(){
		return 'My identification number is:' . $this->identification_number;
	}
}
```

Llenar los atributos de un objeto

Existen tres vías para llenar los atributos de un objeto en ObjectDB:

Insertar los datos manualmente: Útil para modificar uno o pocos atributos del objeto o para objetos con poca cantidad de atributos. No se recomienda su uso, ya que engrandece y ofusca el código resultante.

Mediante el método *setDataFromArray* de la clase *ODBObject*: Permite llenar los atributos del objeto mediante un arreglo de cadenas pasado por parámetro al método *setDataFromArray*. Principalmente se utiliza para crear objetos con información obtenida dinámicamente.

Al usar el método *setDataFromParamsList* de la clase *ODBObject*: Llena los atributos del objeto con los parámetros pasados al método *setDataFromParamsList*. Es cómodo y se aconseja su uso para la mayoría de las situaciones, sin embargo debe tenerse cuidado de insertar la cantidad justa de parámetros y en el orden definido en la tabla.

Sin importar que vía desee utilizarse, nunca debe intentar insertarse/modificarse el atributo llave de la tabla, lo cual podría, en dependencia de la situación, generar un error o incluir comportamiento no deseado. A continuación un código que detalla los tres modos de llenar/modificar los atributos de un objeto.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// crea un nuevo objeto "estudiante", con atributos en blanco
$student = new student();

// cambia los atributos manualamente
$student->identification_number = '3025'; // se permiten cambios porque no es la llave de la tabla
$student->name = 'Salvi Pascual';
$student->age = 37;
$student->preferences = 'color=orange,grossery=pie,serie=futurama';

// cambia los atributos por medio de setDataFromArray
$attributes = Array('3025','Salvi Pascual','24','color=orange,grossery=pie,serie=futurama');
$student->setDataFromArray($attributes);

// cambia los atributos por medio de setDataFromParamsList
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');
```

Enviar una consulta personalizada al servidor

No obstante ObjectDB elimina en alto grado el escribir código en lenguaje SQL, y potencia la utilización de funciones predefinidas que cubren la mayor parte de las necesidades del programador, en ciertas ocasiones en necesario enviar consultas propias al servidor para realizar complicadas búsquedas. Es para ello que se utiliza el método *query* de la clase controladora *objectDB*. En el ejemplo siguiente, primero se obtienen y muestran los diez primeros estudiantes cuya edad es mayor a 25 años y tengan entre sus preferencias el color naranja; luego se elimina el último estudiante cuya serie preferida sea 24 horas (nada personal).

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// carga los 10 primeros estudiantes cuya edad>25 y prefieran el color naranja
$db = new objectDB();
$students_list = $db->query("SELECT * FROM student WHERE age>'25' AND preferences LIKE
'%color=orange%' LIMIT 10");

// imprime la lista de estudiantes
echo 'List of students in database with age>25 and orange as color preference';
for($i=0; $iquery("DELETE FROM student WHERE preferences LIKE '%serie=24 hours%' ORDER BY id_student
DESC LIMIT 1");
```

Nótese que para el caso de consultas tipo SELECT el método devuelve un arreglo de objetos (tal como el método *getObjs*); para consultas sin respuesta, como DELETE, INSERT, etc. el método devuelve NULL.

Modificar objetos salvados

Una vez salvado un objeto en la base de datos, es posible cargarlo y cambiar su valor. Para ello es necesario obtener el objeto mediante la función *getObjs* de la clase controladora *objectDB* (ejemplificada en la sección anterior), cambiar sus valores y guardarlo nuevamente mediante la función *save*, de la clase *ODBObject* (mostrada también en la sección anterior) o mediante la función *saveObj*, de la clase controladora *objectDB*. El código a continuación se realiza dado el uso de esta última forma, no ejemplificada con anterioridad y continúa al ejemplo anterior.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// carga todos los estudiantes guardados
$db = new objectDB();
$students_list = $db->getObjs('student');

// selecciona al primer estudiante escogido
$student = $students_list[0];

// cambia los parámetros de los estudiantes
$student->name = 'Cristobal Colón';
```

```
$student->age = 37;

// guarda al estudiante modificado
$db->saveObj($student);
// $student->save(); // this works too

// muestra al estudiante modificado
echo $student;
```

Obtener el último objeto salvado

Con generalidad es necesario cargar el último objeto almacenado en la base de datos. Dicha funcionalidad es sumamente útil cuando, por ejemplo, es necesario mostrar solo el último comentario de un usuario o noticia más actual. Existen versátiles mecanismos para ejecutar esta acción con ObjectDB, como por ejemplo, cargar todos los elementos de una tabla mediante *getObjs* y recorrer la lista que el método devuelve hasta el final, o (en un escenario menos costoso) ejecutar el método *query* (anteriormente visto) con el código en lenguaje SQL mostrado a continuación:

```
$db = new objectDB();
$result = $db->query("SELECT * FROM student ORDER BY id_student DESC LIMIT 1");
```

Pero para aprovechar realmente los recursos de caché de ObjectDB, y reducir al mínimo los accesos a la base de datos, es preferible utilizar el método *getLastObject*, de la clase controladora *objectDB*.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// carga el ultimo objeto salvado
$db = new objectDB();
$last_student = $db->getLastObject('student');

// imprime al ultimo estudiante modificado
echo $last_student;
```

Eliminar un objeto de la base de datos

Existen dos formas de eliminar un único objeto de la base de datos. La primera es utilizar el método *remove*, propio de la clase *ODBObject* y existente para cualquier objeto que herede de esta. La segunda es mediante el método *removeObj*, de la clase controladora *objectDB*, a la cual hay que entrarle como parámetro el objeto a borrar. A continuación se eliminan los dos primeros estudiantes de la tabla, en caso de ambos existir.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// carga todos los estudiantes en la base de datos
$db = new objectDB();
$students_list = $db->getObjs('student');

// elimina al primer estudiante
$students_list[0]->remove();

// elimina al segundo estudiante
$db->removeObj($students_list[1]);
```

Eliminar múltiples objetos de la base de datos

Al hacer uso las vías mencionadas con anterioridad es posible eliminar un único objeto mediante una referencia al mismo. Esto es ineficiente en caso de querer borrar todas las entradas de una tabla, o muchas entradas que cumplan con una condición dada. Para eso ObjectDB brinda el método *remove*, ubicado en la clase controladora *objectDB*, al cual se le deben dar como parámetros el nombre de la tabla en la cual actuar y opcionalmente una expresión en lenguaje SQL para filtrar la eliminación. En caso de no pasar dicha expresión al método *remove*, se eliminarán todos los objetos de ese tipo almacenados en la base de datos.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";

// elimina a todos los estudiantes cuya edad>32 años
$db = new objectDB();
$db = $db->remove('student',"age>'32'");
```

Relaciones entre tablas

Una de las mayores potencialidades de ObjectDB es la facilidad para crear relaciones entre las tablas de la base de datos sin prácticamente modificar el modelo de tablas y objetos establecido. Entre los objetivos principales de ObjectDB, se encuentra que el programador sea capaz de crear, eliminar y acceder a relaciones entre tablas con pocas líneas de código, y mediante una API sencilla de aprender y utilizar sin poseer altos conocimientos del lenguaje SQL. Detrás de las facilidades de uso para relacionar tablas, se encuentra una vez más el motor de caché de ObjectDB. Esto permite reducir en alto grado accesos al servidor de base de datos y elevar con ello el rendimiento final de las aplicaciones que utilicen la biblioteca.

Crear una tabla de relación

Para relacionar dos tablas de ObjectDB, el primer paso a seguir es crear una tabla de relación (o nexo) entre las dos tablas que se quieran conectar. El nombre de la tabla nexo debe componerse de la cadena *relation*, seguida del caracter underscore (_), luego el nombre de la primera tabla a relacionar, nuevamente el caracter underscore y por último el nombre de la segunda tabla a relacionar. Como ejemplo, el nombre de una tabla nexo entre la tabla *student* y una nueva tabla llamada teacher sería *relation_student_teacher*. El orden de los nombres de las tablas que componen la tabla nexo carece de importancia, por lo cual sería lo mismo escribir *relation_student_teacher* que *relation_teacher_student*.

Las tablas de nexo deben componerse de tres campos:

Llave de la tabla nexo

> Representa la llave de la tabla nexo. Esta cumple iguales requerimientos que el resto de las tablas; debe ser numérica, autoincrementable y de nombre *id_nombretabla*. Como ejemplo, la tabla *relation_student_teacher* tendría de llave el nombre *id_relation_student_teacher*.

Llave de la primera tabla a relacionar

> Este campo no es un atributo llave, sino que tiene igual nombre que la llave de la primera tabla a relacionar. Por ejemplo, para la tabla nexo *relation_student_teacher*, este campo sería *id_student*.

Llave de la segunda tabla a relacionar

> Este campo no es un atributo llave, sino que tiene igual nombre que la llave de la segunda tabla a relacionar. Por ejemplo, para la tabla nexo *relation_student_teacher*, este campo sería *id_teacher*.

El ejemplo a continuación muestra un fragmento de código en lenguaje SQL, optimizado para MySQL v5.x que genera la tabla *student* (utilizada hasta ahora), la nueva tabla *teacher* y la tabla nexo para relacionar ambas.

```
CREATE TABLE student (
      id_student INT(11) KEY AUTO_INCREMENT,
      identification_number VARCHAR(11) NOT NULL,
      name varchar(50) NOT NULL,
      age INT(2) NULL,
      preferences TEXT
);

CREATE TABLE teacher (
      id_teacher INT(11) KEY AUTO_INCREMENT,
      professorship_number VARCHAR(11) NOT NULL,
      name varchar(50) NOT NULL,
      experience_time INT(2) NULL,
      marital_status TINYINT(1),
      last_university_payment_date DATE
);

CREATE TABLE relation_student_teacher (
      id_relation_student_teacher INT(11) KEY AUTO_INCREMENT,
      id_student INT(11),
      id_teacher INT(11)
);
```

A continuación se muestra el contenido para el archivo *teacher.php*, el cual define la clase correspondiente a la tabla con igual nombre. La clase *student*, ubicada en el archivo *student.php* se muestra en secciones anteriores. No es necesario crear una clase que represente la tabla nexo, dado que el proceso de trabajar con relaciones nunca exige instanciar la misma.

```
// constantes para el estado civil
define("SINGLE",  0x01, true);
```

```
define("MARRIED",       0x02, true);
define("DIVORCED",      0x03, true);
define("WIDOWED",       0x04, true);

class teacher extends ODBObject{
      public $professorship_number;
      public $name;
      public $experience_time;
      public $marital_status; // constants
      public $contract_date;  // date
}
```

Definir una relación entre dos objetos

El método *addRelation* de la clase *ODBObject* define una relación entre dos objetos. Al mismo debe pasársele como parámetro un objeto tipo *ODBObject*, existente en la base de datos (debe salvarse antes) y con el cual previamente se haya creado una tabla de relación. En caso que alguno de los objetos que se intenta relacionar no exista en la base de datos, se lanzará una excepción *ODBEObjectNotInDatabase*; como política de ObjectDB no salva objetos sin la petición directa del programador. A continuación se muestra un ejemplo de cómo relacionar dos objetos, de tipo *student* y *teacher*, definidos con anterioridad.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";
require_once "teacher.php";

// crea, establece los valores y guarda un nuevo estudiante
$student = new student();
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');
$student->save();

// crea, establece los valores y guarda un nuevo profesor
$teacher = new teacher();
$teacher->setDataFromParamsList('math201','Stephen W. Hawking',37,MARRIED,date("Y-m-d"));
$teacher->save();

// crea una relación entre
$student->addRelation($teacher); // al igual que: $teacher->addRelation($student);
```

Dado que un estudiante está relacionado con su profesor de igual modo que el profesor con su estudiante, la línea *$student->addRelation($teacher);* puede cambiarse por *$teacher->addRelation($student);* y obtener el mismo resultado. Será lanzada una excepción si para dos objetos, una relación es salvada más de una vez.

Obtener las relaciones para un objeto

Una vez creadas relaciones entre objetos, es necesario aprender como consultarlas fácilmente. Para ello *ObjectDB* ofrece el método *relations*, perteneciente a la clase *ODBObject*. El mismo recibe como parámetro una cadena de texto, que representa el nombre de la clase a consultar, y devuelve la lista de objetos relacionados con el objeto desde el cual se consultó. En el ejemplo a continuación se muestra este funcionamiento.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";
require_once "teacher.php";

// crea, establece los valores y guarda un nuevo estudiante
$student = new student();
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');
$student->save();

// valores de los profesores de la universidad
$teachers_values = Array(
      Array('math201','Stephen W. Hawking',37,MARRIED,date("Y-m-d")),
      Array('math201','Albert Einstain',68,MARRIED,date("Y-m-d")),
      Array('math201','Nicolas Coppernico',37,WIDOWED,date("Y-m-d")),
      Array('math201','Felix Varela',37,SINGLE,date("Y-m-d")),
      Array('math201','Meredick Jhones',37,DIVORCED,date("Y-m-d"))
);

// crea y relaciona a los "profesores"
$teachers = Array(count($teachers_values));
for ($i=0; $isetDataFromArray($teachers_values[$i]);
      $teachers[$i]->save();
      // relation each teacher
      $student->addRelation($teachers[$i]);
}
```

```
//carga y muestra todas las relaciones
$teachers = $student->relations('teacher');
foreach ($teachers as $teacher) echo $teacher;
```

Conocer si dos objetos están relacionados

En ocasiones es útil conocer si dos objetos se encuentran relacionados entre sí, para ello la clase *ODBObject* brinda el método *isRelationedWith*, al cual se pasa como parámetro un objeto tipo *ODBObject* y devuelve *true* en caso de existir la relación y *false* en caso contrario. Al igual que *addRelation*, este método lanzará una excepción *ODBEObjectNotInDatabase* si alguno de los objetos no se encuentran en la base de datos. A continuación se muestra un ejemplo de cómo trabajar con el método *isRelationedWith*.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";
require_once "teacher.php";

// crea, establece los valores y guarda un nuevo estudiante
$student = new student();
$student->setDataFromParamsList(3025,'Salvi Pascual',24,'color=orange,grossery=pie,serie=futurama');
$student->save();

// crea, establece los valores y guarda un nuevo profesor
$teacher = new teacher();
$teacher->setDataFromParamsList('math201','Stephen W. Hawking',37,MARRIED,date("Y-m-d"));
$teacher->save();

$student->isRelationedWith($teacher); // esto retornara falso

$student->addRelation($teacher);
$student->isRelationedWith($teacher); // esto retornara verdadero
```

Es posible utilizar el método *relations* para obtener la lista de profesores relacionados con un estudiante específico y luego recorrer ésta preguntando si alguno es el profesor que buscamos, pero esta solución incrementa innecesariamente la cantidad de líneas de código y no aprovecha al máximo el motor de cache de *ObjectDB*, por lo cual se aconseja no desarrollar esta

solución. Al igual que al usar el método *addRelation*, tiene el mismo resultado escribir *$student->addRelation($teacher);* que *$teacher->addRelation($student);*.

Eliminar una relación entre dos objetos

El método *removeRelation* de la clase *ODBObject* se utiliza para eliminar una relación existente entre dos objetos. Al mismo se le pasa como parámetro un objeto tipo *ODBObject* con el cual se desea desconectar el objeto en uso. Este método lanzará una excepción *ODBEObjectNotInDatabase* si alguno de los objetos no se encuentran en la base de datos. También lanzará una excepción tipo *ODBENoRelationBetween* si no existe relación entre ambos objetos. El siguiente ejemplo muestra como se puede eliminar una relación mediante el uso de *removeRelation*. En el ejemplo a continuación provoca igual resultado escribir *$student->removeRelation($teacher);* que *$teacher->removeRelation($student);*.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "student.php";
require_once "teacher.php";

// carga el ultimo estudiante guardado
$db = new objectDB();
$student = $db->getLastObject('student');

// carga la primera relacion del estudiante cargado
$teachers = $student->relations('teacher');
$teacher = $teachers[0];

// elimina la relacion
$student->removeRelation($teacher); // same than: $teacher->removeRelation($student);
```

Destruir todas las relaciones de un objeto

En diversas ocasiones es necesario desposeer a un objeto de todas sus relaciones. Ejemplos claros de esta necesidad son cuando un cliente compra todas las existencias de un producto en una tienda online, o cuando antes de despedir a un profesor se desean eliminar todas las relaciones que el mismo posea con sus estudiantes. El método *removeAllRelations* realiza este trabajo. Del último caso descrito puede verse un ejemplo en el código a continuación.

```
require_once "../objectDB/objectDB-mysql-v3.0.php"; // primero esto incluirá
require_once "teacher.php";

// carga el ultimo profesor en evaluacion, para despedirlo
$db = new objectDB();
$teacher = $db->getLastObject('teacher');

// elimina todas las relaciones con sus antiguos estudiantes
$teacher->removeAllRelations('student');
```

Al método *removeAllRelations* se le pasa como parámetro una cadena de texto que representa el nombre de la tabla con la cual eliminar todas las relaciones. Este método lanzará una excepción *ODBEObjectNotInDatabase* si el objeto no se encuentra en la base de datos o no ha sido salvado aún. También lanzará una excepción tipo *ODBETableNotDefined* si la cadena de texto insertada como parámetro no representa el nombre de alguna tabla de la base de datos.

Programa de aplicación de interfaz[1]

Esta sección explica el programa de aplicación de interfaz, dividiendo la explicación en cuatro clases que componen la clase ObjectDB. La tabla que se muestra a continuación menciona cada una de esas clases y define brevemente las responsabilidades de cada una.

Clase	Descripción
ODBException	Para el control de excepciones en la clase ObjectDB
ODBConnection	Configuración de la conexión y el trabajo a bajo nivel
ODBObject	Contiene las clases de información y les proporciona la funcionalidad de ObjectDB
ObjectDB	Control de clase. Principal funcionalidad de la biblioteca

[1] Se conoce comúnmente por el término en inglés API (*Application Programming Interface*)

ODBException

La clase mostrada a continuación define al objeto *ODBException*, el mismo crea una base para el trabajo con excepciones propias de *ObjectDB*.

```
class ODBException extends Exception {
      public $message;

      function ODBException($message=null);
      public function __toString();
}
```

Del objeto *ODBException* heredan las clases de excepción listadas a continuación. Estas definen la comunicación de *ObjectDB* con el programador, y se levantan en casos específicos, explicados en próximas secciones. En caso de realizar cambios a *ObjectDB*, es un error lanzar objetos tipo *ODBException*, en vez de esto los programadores deben crear sus propias excepciones que hereden de *ODBException*.

```
class ODBENotConnected        extends ODBException{}  // lanzada si hay problemas con la conexion a
la base de datos
class ODBEConnectionNotActive extends ODBException{}  // lanzada si la conexion se cerró previamente
class ODBEObjectNotInDatabase extends ODBException{}  // lanzada si el objeto a tratar no existe en
la base de datos
class ODBEWrongSQLSentence    extends ODBException{}  // lanzada si la sentencia SQL no es correcta
class ODBETableNotDefined     extends ODBException{}  // lanzada si se trata de usar una tabla no
definida
class ODBEClassNotDefined     extends ODBException{}  // lanzada si se trata de usar una clase no
definida
class ODBECannotChangeKey     extends ODBException{}  // lanzada si se trata de cambiar la llave de
una fila
class ODBEObjectNotExist      extends ODBException{}  // lanzada si se trata de usar un objeto
borrado o que no existe en la base de datos
class ODBEMoreThanOneInstance extends ODBException{}  // lanzada si se instancia una clase singletton
mas de una vez
class ODBERelationAlreadyExist extends ODBException{} // lanzada si se trata de establacer una
relacion previamente definida
```

```
class ODBENoRelationBetween   extends ODBException{}  // lanzada si se trata de borrar una relacion
que no existe
```

Constructor

Esta clase solo debe construirse mediante otras que hereden de ella. Al constructor debe pasársele como parámetro una cadena de texto con un mensaje personalizado para ser mostrado al lanzarse la excepción. En caso de no pasarse mensaje alguno, al lanzar la excepción se mostrará el error por defecto que devuelva la base de datos.

```
Parámetros
String | none: mensaje para mostrar cuando la excepción sea lanzada.
Excepciones
none
```

message

El atributo *message* guarda una cadena de texto que contiene un texto personalizado para ser mostrado al lanzar la excepción. Este atributo es público, por lo cual puede ser modificado o consultado en cualquier momento después de ser creado el objeto, pero por comodidad puede ser inicializado también en el constructor de la clase.

```
Tipo de dato
String
Banderas
public
```

__toString()

Este método genera una cadena de texto con información acerca del contenido de la clase. Puede ser llamado literalmente mediante la expresión *$obj->__toString();*, pero está preparado para ejecutarse al tratar el objeto como una cadena de texto, por lo cual es más sencillo llamarlo mediante expresiones como: *echo $obj;* ó *throw $obj;*.

```
Tipo de retorno
```

```
String
Parámetros
none
Excepciones
none
```

ODBConnection

La clase a continuación define al objeto *ODBConnection*, el mismo abre y destruye conexiones con la base de datos e implementa las funciones básicas de acceso a la misma. No se aconseja utilizar las funciones de conexión directamente desde un objeto *ODBConnection*, en vez de ello sería mejor acceder a métodos más refinados, brindados por los objetos *ODBObject* y *objectDB*, con el fin de asegurar el trabajo con la caché.

Un objeto *ODBConnection* nunca debería ser creado por el usuario, a pesar que no se limita la posibilidad de hacerlo. Este objeto es instanciado y utilizado por el resto de las clases de *objectDB* para realizar accesos a la base de datos. El único roce del programador con este objeto debería ser para modificar los parámetros de configuración.

```
class ODBConnection {
      /* start - configure */
      static public $host   = "localhost";
      static public $port   = "3306"; // mysql default port
      static public $dbname = "database_name";
      static public $user   = "database_username";
      static public $pass   = "database_password";
      /* end - configure */

      public $connect = null;

      public function ODBConnection();
      public function query($sql);
      public function getTableKey($tbName);
      public function getRelationTableName($tbNameA,$tbNameB);
      public function getLastObject($tbName);
      public function testTable($tbName);
      public function close();
      public function __destruct();
}
```

Constructor

Esta clase solo deben construirla otros objetos de la biblioteca, aunque no se restringe su instanciación y uso por el programador. Al ser construida abre una conexión con la base de datos, dado lo insertado en los parámetros de configuración, y la cierra al ser destruida. En caso de estar incorrectos los parámetros de configuración lanza una excepción de tipo *ODBENotConnected*.

```
Parámetros
none
Excepciones
ODBENotConnected: se lanza si hay problemas con la conexión
```

Destructor

Cierra la conexión cuando el objeto es destruido.

```
Excepciones
ODBEConnectionNotActive: es lanzada si la conexión fue cerrada con anterioridad.
```

host

El atributo *host* guarda una cadena de texto que representa la dirección del servidor de base de datos a conectarse. Puede ser especificado mediante el nombre de una máquina o una dirección IP.

```
Tipo de dato
String
Banderas
public
static
```

port

Especifica el puerto por el cual el servidor de base de datos escucha.

```
Tipo de dato
Integer
Banderas
public
static
```

dbname

Constituye una cadena de texto que representa el nombre de la base de datos en el servidor.

```
Tipo de dato
String
Banderas
public
static
```

user

El atributo *user* guarda una cadena de texto que representa el nombre de usuario para acceder al servidor de base de datos.

```
Tipo de dato
String
Banderas
public
static
```

pass

Guarda una cadena de texto que representa la contraseña de usuario para acceder al servidor de base de datos.

```
Tipo de dato
String
Banderas
public
static
```

connect

El atributo *connect* guarda el token de conexión con la base de datos. Este valor en situaciones normales nunca debería ser utilizado, sin embargo se mantiene con visibilidad pública para casos en que el programador desee combinar *objectDB* con las funciones nativas de la base de datos en la cual trabaja.

```
Tipo de dato
Integer
Banderas
public
```

query()

Realiza una consulta a la base de datos, en caso de ser una consulta tipo SELECT, devuelve un arreglo de objetos *ODBObject*, en caso contrario devuelve null. Se le pasa como parámetro una cadena de texto que representa una consulta válida en lenguaje SQL. Lanza una excepción de tipo *ODBEWrongSQLSentence* en caso que se le pase una consulta incorrecta.

```
Tipo de retorno
Array de ODBObject
Parámetros
String: una consulta válida en lenguaje SQL
Excepciones
ODBEWrongSQLSentence: lanzada si el parámetro no es una consulta válida
```

getTableKey()

Devuelve la llave de la tabla para el nombre de tabla pasado por parámetro.

```
Tipo de retorno
String: nombre del campo que funciona como llave de la tabla
Parámetros
String: nombre de una tabla
Excepciones
none
```

getRelationTableName()

Devuelve el nombre de la tabla que relaciona dos tablas. Se le pasan como parámetro dos nombres de tablas existentes en la base de datos, y devuelve una excepción de tipo *ODBETableNotDefined* si no existe una tabla de relación para ellas.

```
Tipo de retorno
String: nombre de la tabla de relación
Parámetros
String: nombre de una tabla en la base de datos
String: nombre de otra tabla en la base de datos
Excepciones
ODBETableNotDefined: es lanzada si la tabla de relaciones no existe en la base de datos
```

getLastObject()

Obtiene el último objeto salvado en una tabla. Se le pasa como parámetro el nombre de la tabla para obtener su último objeto salvado. Lanza una excepción de tipo *ODBETableNotDefined* si la tabla pasada por parámetros no existe.

```
Tipo de retorno
ODBObject
Parámetros
String: nombre de una tabla en la base de datos
Excepciones
ODBETableNotDefined: es lanzada si la tabla pasada por parámetros no existe
```

testTable()

Prueba si una tabla existe en la base de datos. Se le pasa como parámetro el nombre de la tabla a probar.

```
Tipo de retorno
Boolean: TRUE si la tabla existe, FALSE en caso contrario
Parámetros
String: nombre de una tabla en la base de datos
Excepciones
none
```

close()

Cierra una conexión con la base de datos. Lanza una excepción de tipo *ODBEConnectionNotActive* si la conexión fue cerrada con anterioridad.

```
Tipo de retorno
none
Parámetros
none
Excepciones
ODBEConnectionNotActive: es lanzada si la conexión fue cerrada con anterioridad.
```

ODBObject

La clase mostrada a continuación define al objeto *ODBObject*, el mismo almacena la información de una fila de la base de datos, y provee al programador de funcionalidades que le permiten interactuar con la misma. Herede de esta clase para crear incrementar su funcionalidad y nunca cree nuevas instancias manualmente, de esto se encargan los métodos de la biblioteca.

```
class ODBObject {
      public function setId($id=null);
      public function setSaveStatus();
      public function setDataFromArray($data);
      public function setDataFromParamsList();
      public function getId();
      public function getState();
      public function getTableName();
      public function getTableKeyName();
      public function isSaved();
      public function save();
      public function remove();
      public function isRelationedWith($object);
      public function relations($tbName);
      public function addRelation($object);
      public function removeRelation($object);
      public function removeAllRelations($tbName);
      public function __toString();
}
```

Las siguientes constantes representan estados que un objeto puede tener.

```
define("OBJECT_SAVED", 0x01, true); // define un ODBObject guardado en la base de datos
define("OBJECT_NOT_SAVED", 0x02, true);   // define un ODBObject no guardado en la base de datos
define("OBJECT_DELETED", 0x03, true);     // define un ODBObject eliminado con la funcion remove()
```

setId()

Establece la llave para una fila de la tabla, pero sólo si el objeto no ha sido salvado.

```
Tipo de retorno
none
Parámetros
String: llave de la tabla
Excepciones
ODBECannotChangeKey: Lanzada si se intenta cambiar la llave a un objeto ya salvado.
```

setSaveStatus()

Marca un objeto como salvado; un objeto guardado no se puede volver a marcar. Este método es usado para el trabajo interno de ObjectDB, y solo se ha puesto público para que la clase *objectDB* tenga visibilidad. Al salvar un objeto mediante las funciones correspondientes, la biblioteca se encarga automáticamente de marcarlo, por lo cual no se aconseja al programador usar el mismo.

```
Tipo de retorno
none
Parámetros
none
Excepciones
none
```

setDataFromArray()

Llena un objeto mediante una matriz de información.

```
Tipo de retorno
none
Parámetros
Array; valores del objeto ordenados según los campos de la base de datos.
Excepciones
```

```
ODBEObjectNotExist: Lanzada si se intenta llenar un objeto eliminado o inexistente.
```

setDataFromParamsList()

Llena un objeto con una lista de parámetros.

```
Tipo de retorno
none
Parámetros
Tantos parámetros como columnas tenga la base de datos, sin contar la llave de la tabla.
Excepciones
ODBEObjectNotExist: Lanzada si se intenta llenar un objeto eliminado o inexistente.
```

getId()

Obtiene la llave de la tabla para esta fila.

```
Tipo de retorno
String: la tabla para esta fila
Parámetros
none
Excepciones
none
```

getState()

Obtiene el estado de un objeto.

```
Tipo de retorno
Integer: constantes OBJECT_SAVED, OBJECT_NOT_SAVED, OBJECT_DELETED
Parámetros
none
Excepciones
```

```
none
```

getTableName()

Devuelve el nombre de la tabla en la cual se guarda este objeto.

```
Tipo de retorno
String: nombre de la tabla
Parámetros
none
Excepciones
none
```

getTableKeyName()

Devuelve el nombre del campo que representa la llave de la tabla.

```
Tipo de retorno
String: nombre del campo llave en la tabla
Parámetros
none
Excepciones
none
```

isSaved()

Comprueba si el objeto está en la base de datos.

```
Tipo de retorno
Boolean: TRUE si el objeto está en la base de datos, FALSE en caso contrario
Parámetros
none
Excepciones
```

```
none
```

save()

Si el objeto acaba de ser creado, lo almacena en la base de datos, si el objeto ya estaba en la base de datos y fue modificado, lo actualiza.

```
Tipo de retorno
none
Parámetros
none
Excepciones
ODBEObjectNotExist: Lanzada si el objeto ha sido eliminado o no existe
```

remove()

Elimina un objeto de la base de datos de manera permanente.

```
Tipo de retorno
Boolean: TRUE si el objeto fue eliminado, se lanza una excepción en caso contrario.
Parámetros
none
Excepciones
ODBEObjectNotInDatabase: Lanzado si el objeto no está en la base de datos ODBEObjectNotExist: Lanzada
si el objeto fue eliminado o no existe
```

isRelationedWith()

Comprueba si existe relación entre dos objetos.

```
Tipo de retorno
Boolean: TRUE si existe una relación, FALSE en caso contrario
Parámetros
```

```
ODBObject: Objeto con el cual probar si existe una relación.
Excepciones
none
```

relations()

Obtiene las relaciones que tiene un objeto con otro

```
Tipo de retorno
Array: lista de objetos relacionados con este
Parámetros
String: nombre de la tabla con la cual se quieren obtener relaciones
Excepciones
ODBEObjectNotInDatabase: Lanzada si este objeto no existe en la base de datos ODBETableNotDefined:
Lanzada si la tabla de relaciones no existe
ODBEObjectNotExist: Lanzada si el objeto fue eliminado o no existe
```

addRelation()

Define una relación entre dos objetos

```
Tipo de retorno
none
Parámetros
ODBObject: Objeto con el cual definir la relación
Excepciones
ODBEObjectNotInDatabase: Lanzada si este objeto no existe en la base de datos ODBETableNotDefined:
Lanzada si la tabla de relaciones no existe
ODBEObjectNotExist: Lanzada si el objeto fue eliminado o no existe
ODBERelationAlreadyExist: lanzada si la relación fue previamente definida
```

removeRelation()

Elimina una relación entre dos objetos.

```
Tipo de retorno
none
Parámetros
ODBObject: Objeto con el cual eliminar la relación
Excepciones
ODBEObjectNotInDatabase: Lanzada si este objeto no existe en la base de datos ODBETableNotDefined:
Lanzada si la tabla de relaciones no existe
ODBEObjectNotExist: Lanzada si el objeto fue eliminado o no existe
ODBENoRelationBetween: Lanzada si no existe una relación
```

removeAllRelations()

Elimina todas las relaciones que pueda tener un objeto con una tabla dada.

```
Tipo de retorno
none
Parámetros
String: nombre de la tabla con la cual eliminar todas las relaciones
Excepciones
ODBEObjectNotInDatabase: Lanzada si este objeto no existe en la base de datos ODBETableNotDefined:
Lanzada si la tabla de relaciones no existe
ODBEObjectNotExist: Lanzada si el objeto fue eliminado o no existe
```

__toString()

Este método genera una cadena de texto con información acerca del contenido de la clase. Puede ser llamado literalmente mediante la expresión *$obj->__toString();*, pero está preparado para ejecutarse al tratar el objeto como una cadena de texto, por lo cual es más sencillo llamarlo mediante expresiones como: *echo $obj;* ó *throw $obj;*.

```
Tipo de retorno
String
Parámetros
none
Excepciones
none
```

objectDB

La clase mostrada a continuación define al objeto *objectDB*, este ejecuta las funciones principales para la biblioteca. No instancie esta clase más de una vez, o será generada una excepción de tipo *ODBEMoreThanOneInstance*.

```
class objectDB{
      public function objectDB();
      public function query($sql);
      public function getObjs($tbName,$where="");
      public function saveObj($object);
      public function removeObj($object);
      public function remove($tbName,$where="");
      public function getLastObject($tbName);
}
```

Constructor

Esta clase sigue el patrón de diseño Singletton, por lo cual no debe instanciarse más de una vez. Generará una excepción de tipo *ODBEMoreThanOneInstance* en otro caso.

```
Parámetros
none
Excepciones
ODBEMoreThanOneInstance: Lanzada si se intenta construir más de una instancia
```

query()

Ejecuta una consulta, si es de tipo SELECT, devuelve una lista de objetos, en caso contrario devuelve NULL

```
Tipo de retorno
Array | NULL
```

```
Parámetros
String: una consulta válida
Excepciones
ODBEWrongSQLSentence: Lanzada si la sentencia SQL es incorrecta
```

getObjs()

Obtiene una lista de objetos dado el nombre de la tabla y una expresión SQL para filtrarlos. Deje en blanco el segundo parámetro para obtener toda la tabla.

```
Tipo de retorno
Array: lista de objetos
Parámetros
String: nombre de la tabla de la cual se desea sacar los objetos
String: expresión en lenguaje SQL para filtrar los resultados (pe: id_table='24')
Excepciones
ODBETableNotDefined: Lanzada si se trata de usar una tabla no definida
ODBEWrongSQLSentence: Lanzada si la sentencia SQL es incorrecta
```

saveObj()

Guarda o reemplaza un objeto en la base de datos

```
Tipo de retorno
none
Parámetros
ODBObject: objeto no salvado para guardar o reemplazar
Excepciones
ODBEObjectNotExist: Lanzada si se usa un objeto eliminado o no existente
```

removeObj()

Elimina un objeto de la base de datos

```
Tipo de retorno
Boolean: TRUE si lo elimina, lanza excepciones en otro caso
Parámetros
ODBObject: objecto a eliminar
Excepciones
ODBEObjectNotInDatabase: Lanzada si el objeto a eliminar no existe en la base de datos
ODBEObjectNotExist: Lanzada si se usa un objeto eliminado o no existente
```

remove()

Elimina una lista de objetos dado el nombre de su tabla y una expresión SQL para filtrarlos.

```
Tipo de retorno
none
Parámetros
String: nombre de la tabla de la cual se desea eliminar los objetos
String: expresión en lenguaje SQL para filtrar los resultados (pe: id_table<='24')
Excepciones
ODBETableNotDefined: Lanzada si se trata de usar una tabla no definida
ODBEWrongSQLSentence: Lanzada si la sentencia SQL es incorrecta
```

getLastObject()

Obtiene el último objeto salvado en una tabla.

```
Tipo de retorno
ODBObject: último objeto salvado
Parámetros
String: nombre de la tabla de la cual obtener el último objeto salvado
Excepciones
ODBETableNotDefined: Lanzada si se trata de usar una tabla no definida
```