

**Jefri Maruli**

# Back-End Development

## Materi 4



# Agenda Belajar

- PDO (PHP Data Objects)
- CRUD (Create Read Update Delete)
- API (Application Programming Interface)



# PDO (PHP Data Objects)

**PDO** (PHP Data Objects) adalah interface universal yang disediakan PHP yang dibangun dengan bahasa C untuk berkomunikasi dengan database server berbasis OOP (Object Oriented Programming).



# Konsep PDO



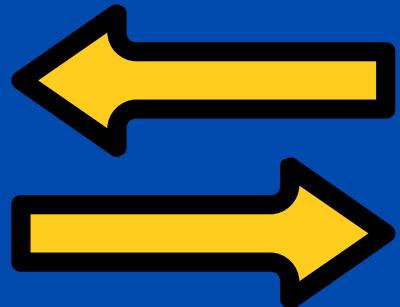
Konsep PDO memiliki tahap untuk akses ke database yaitu melalui interface yang disediakan oleh PHP dan menyediakan metode akses yang seragam ke beberapa database.

# Keunggulan Menggunakan PDO



## Portability / Multi Driver

PDO support dengan banyak database, jadi cocok untuk migration database. Database yang didukung oleh PDO antara lain **MySQL**, **PostgreSQL**, **Oracle**, **SQLite**, **IBM**, **Firebird**, **DBLib**, dan lain-lain



## Flexibility / Easy to Switch

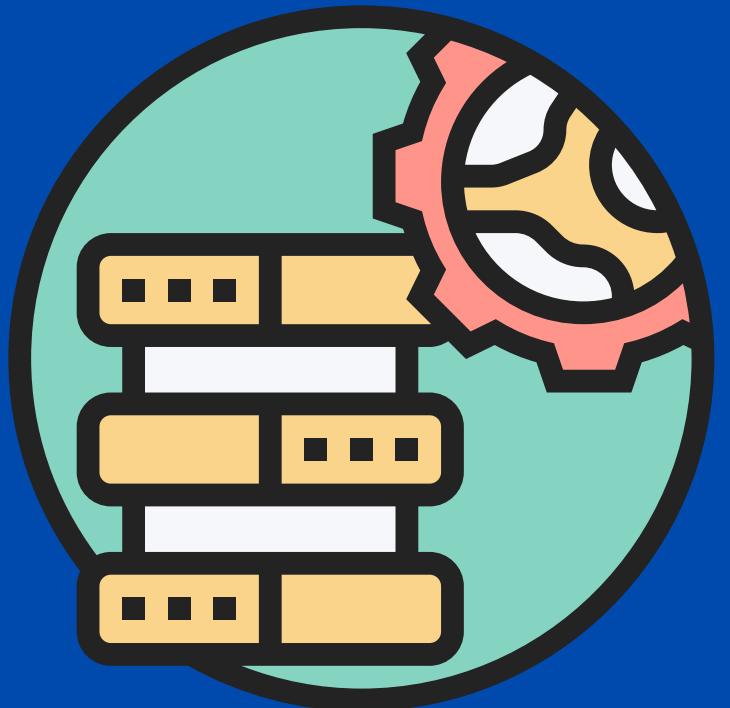
Mudah dalam penggantian database engine yang digunakan karena PDO query setiap database engine tetap sama yang membedakan hanya koneksi saja.

# Kekurangan Menggunakan PDO



## Object Oriented Based

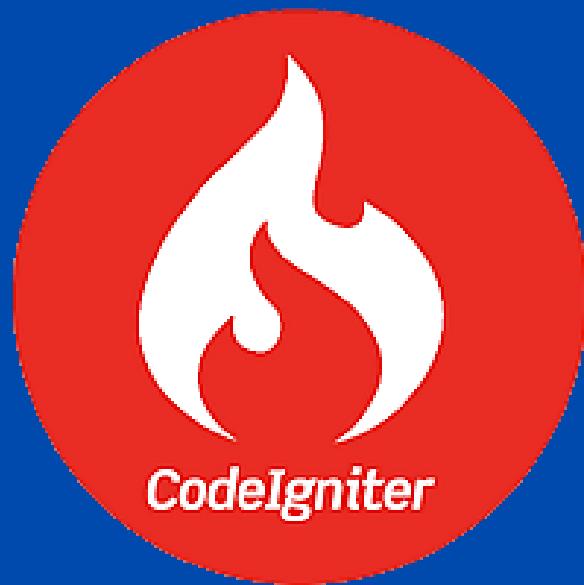
PDO hanya bisa digunakan untuk paradigma Object Oriented Programming (OOP) saja, tidak bisa menggunakan paradigma Procedural Programming (PP) atau Functional Programming (FP).



# Framework Menggunakan PDO



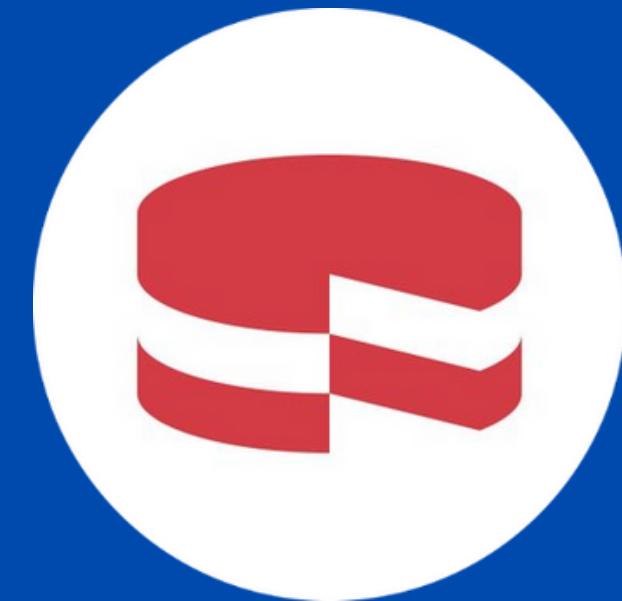
Laravel



CodeIgniter



Yii



Cake PHP

Hampir semua framework terutama yang menggunakan paradigma OOP juga mengimplementasikan PDO sebagai komunikasi database.

# Perbandingan Penerapan Koneksi

Penerapan koneksi ke database dengan PDO dengan Mysql untuk versi PHP <= 5.5 atau Mysqli (Mysql improved) untuk versi PHP >= 5.5.



PDO

```
1 <?php  
2  
3 $pdo = new PDO("mysql:host=localhost; dbname=database", 'username', 'password');
```



Mysqli Procedural

```
1 <?php  
2  
3 $mysqli = mysqli_connect('localhost', 'username', 'password', 'database');
```



Mysqli OOP

```
1 <?php  
2  
3 $mysqli = new mysqli('localhost', 'username', 'password', 'database');
```

# Class Synopsis PDO

```
● ● ●  
1 <?php  
2  
3 class PDO {  
4 /* Methods */  
5 public __construct(  
6     string $dsn,  
7     ?string $username = null,  
8     ?string $password = null,  
9     ?array $options = null  
10 )  
11 public beginTransaction(): bool  
12 public commit(): bool  
13 public errorCode(): ?string  
14 public errorInfo(): array  
15 public exec(string $statement): int|false  
16 public getAttribute(int $attribute): mixed  
17 public static getAvailableDrivers(): array  
18 public inTransaction(): bool  
19 public lastInsertId(?string $name = null): string|false  
20 public prepare(string $query, array $options = []): PDOStatement|false  
21 public query(string $query, ?int $fetchMode = null): PDOStatement|false  
22 public query(string $query, ?int $fetchMode = PDO::FETCH_COLUMN, int $colno): PDOStatement|false  
23 public query(  
24     string $query,  
25     ?int $fetchMode = PDO::FETCH_CLASS,  
26     string $classname,  
27     array $constructorArgs  
28 ): PDOStatement|false  
29 public query(string $query, ?int $fetchMode = PDO::FETCH_INTO, object $object): PDOStatement|false  
30 public quote(string $string, int $type = PDO::PARAM_STR): string|false  
31 public rollBack(): bool  
32 public setAttribute(int $attribute, mixed $value): bool  
33 }
```

```
● ● ●  
1 <?php  
2  
3 $pdo = new PDO("mys
```

Saat pembuatan object (instance) dari class PDO, gambaran umum class yang telah disediakan oleh PHP itu sendiri seperti yang ada di gambar.

# Driver PDO

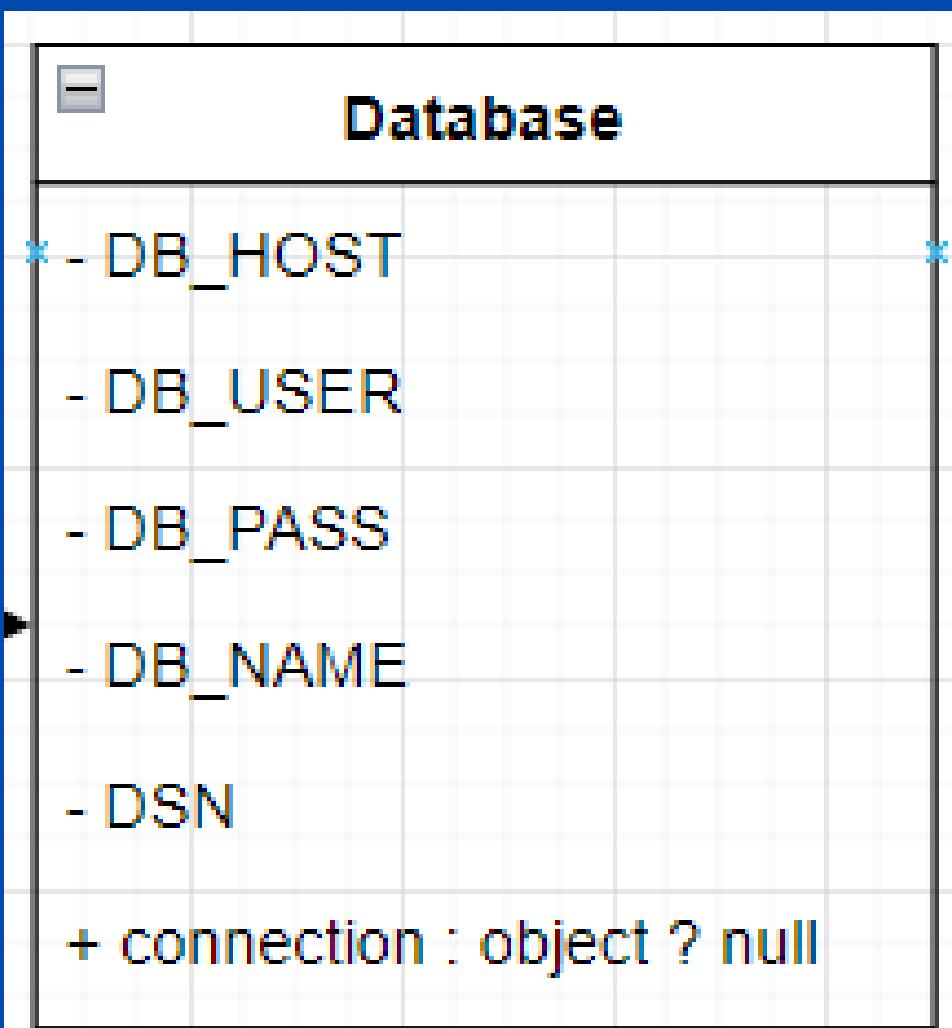
Untuk melihat driver database apa saja yang telah aktif dan bisa langsung digunakan, bisa menggunakan **static function** (Materi OOP), secara default driver yang bisa digunakan oleh PDO yaitu **Mysql**

```
1 <?php  
2  
3 // function yang digunakan getAvailableDrivers adalah statis  
4 // sehingga tidak perlu membuat objek baru  
5 print_r(PDO::getAvailableDrivers());  
6
```

```
<?php  
public static function getAvailableDrivers(): array { }
```

Array ([0] => mysql)

# Class Diagram Koneksi PDO



Buatlah file dengan nama **Database.php** dan buat class **Database** dengan properties yang dibutuhkan seperti:

- Private constant DB\_HOST (hostname)
- Private constant DB\_USER (username mysql)
- Private DB\_PASS (password mysql)
- Private constant DB\_NAME (nama database yang akan dibuat)
- Private constant DSN (Data Source Name yang berisi dari driver, host dan nama database yang akan disatukan jadi satu source data)
- Public variable connection yang digunakan untuk menghubungkan dari PDO ke database server

# Properties Class Database Koneksi PDO

```
● ● ●  
1 <?php  
2  
3 class Database  
4 {  
5     private const DB_HOST = 'localhost';  
6     private const DB_USER = 'root';  
7     private const DB_PASS = '';  
8     private const DB_NAME = 'pdo';  
9     private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';  
10    public ?object $connection = null;  
11 }
```

# Constructor Koneksi PDO

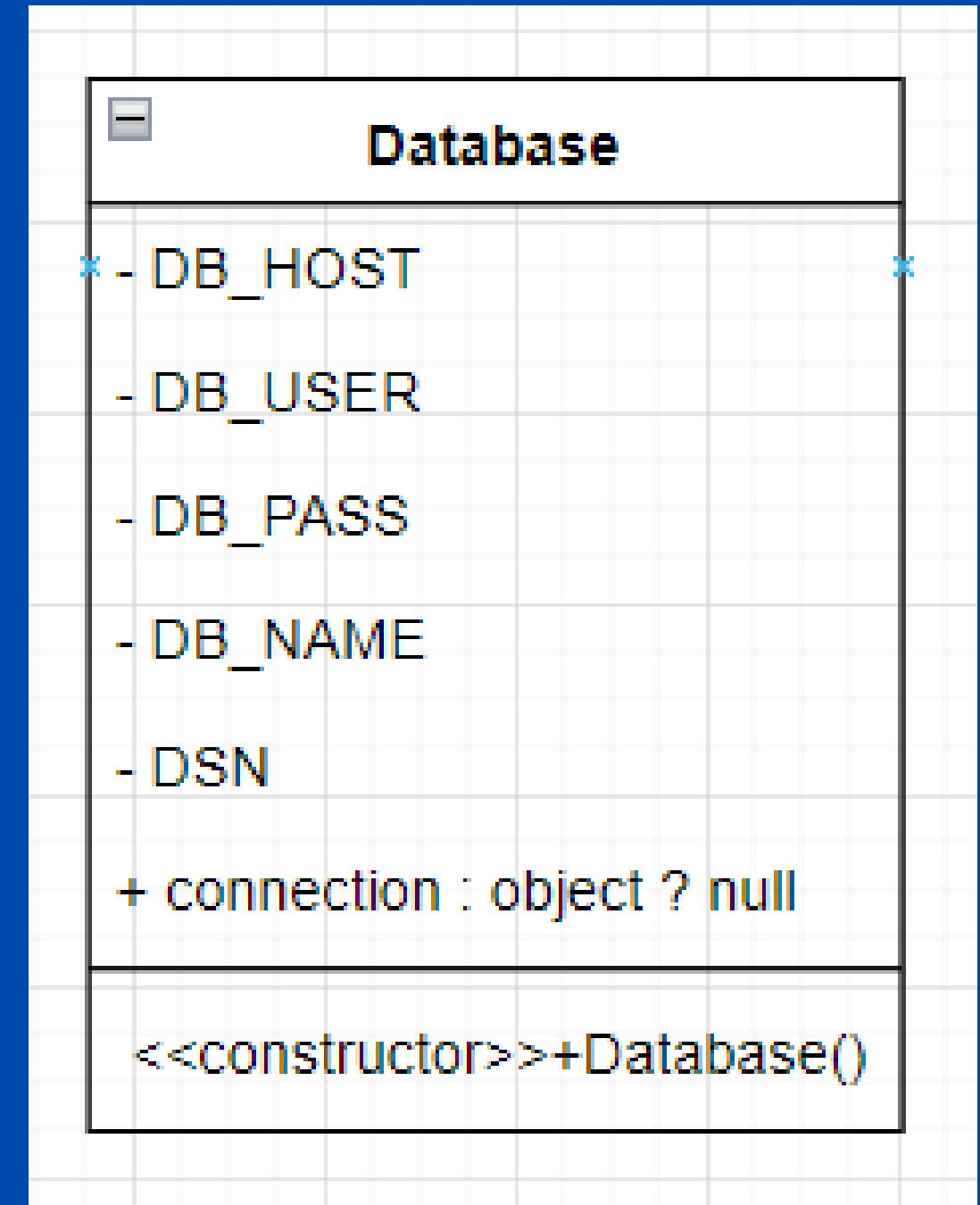
```
● ● ●  
1 public function __construct()  
2 {  
3     try {  
4         $this->connection = new PDO(  
5             self::DSN,  
6             self::DB_USER,  
7             self::DB_PASS  
8         );  
9         $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
10    } catch (PDOException $e) {  
11        echo $e->getMessage();  
12    }  
13    return $this->connection;  
14 }
```

: <<constructor>>+Database()

Buatlah constructor untuk menggunakan property connection dengan **this** keyword, digunakan untuk membuat object dari class PDO, Menggunakan control statement (try & catch) untuk debugging apakah koneksi database sudah sesuai atau belum.

# Class Database

```
1 <?php
2
3 class Database
4 {
5     private const DB_HOST = 'localhost';
6     private const DB_USER = 'root';
7     private const DB_PASS = '';
8     private const DB_NAME = 'pdo';
9     private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';
10    public ?object $connection = null;
11
12    public function __construct()
13    {
14        try {
15            $this->connection = new PDO(
16                self::DSN,
17                self::DB_USER,
18                self::DB_PASS
19            );
20            $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
21        } catch (PDOException $e) {
22            echo $e->getMessage();
23        }
24        return $this->connection;
25    }
26 }
```



# Membuat Object Untuk Testing Koneksi

```
1 <?php
2
3 class Database
4 {
5     private const DB_HOST = 'localhost';
6     private const DB_USER = 'root';
7     private const DB_PASS = '';
8     private const DB_NAME = 'pdo';
9     private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';
10    public ?object $connection = null;
11
12    public function __construct()
13    {
14        try {
15            $this→connection = new PDO(
16                self::DSN,
17                self::DB_USER,
18                self::DB_PASS
19            );
20            $this→connection→setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
21        } catch (PDOException $e) {
22            echo $e→getMessage();
23        }
24        return $this→connection;
25    }
26}
27
28 $database = new Database();
29
```

Constructor akan bekerja otomatis dengan cara membuat object dari class Database.

# Testing Koneksi Database

```
● ● ●  
1 public function __construct()  
2 {  
3     try {  
4         $this->connection = new PDO(  
5             self::DSN,  
6             self::DB_USER,  
7             self::DB_PASS  
8         );  
9         $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
10    } catch (PDOException $e) {  
11        echo $e->getMessage();  
12    }  
13    return $this->connection;  
14 }
```



SQLSTATE[HY000] [1049] Unknown database 'pdo'

Saat dijalankan jika konfigurasi database sudah sesuai maka akan menjalankan block code **try**, jika tidak maka akan menjalankan block code **catch** dan menampilkan pesan error sesuai dengan kesalahan konfigurasinya masing-masing.

# Kumpulan Error Koneksi Database

Kesalahan saat mengisi konfigurasi hostname

```
SQLSTATE[HY000] [2002] php_network_getaddresses: getaddrinfo failed: No such host is known.
```

```
● ● ●  
1 private const DB_HOST = 'localhost';
```

Kesalahan saat mengisi konfigurasi untuk username ketika tidak menggunakan password

```
SQLSTATE[HY000] [1045] Access denied for user 'rot'@'localhost' (using password: NO)
```

```
● ● ●  
1 private const DB_USER = 'rot';
```

Kesalahan saat mengisi konfigurasi untuk password ketika menggunakan password

```
SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost' (using password: YES)
```

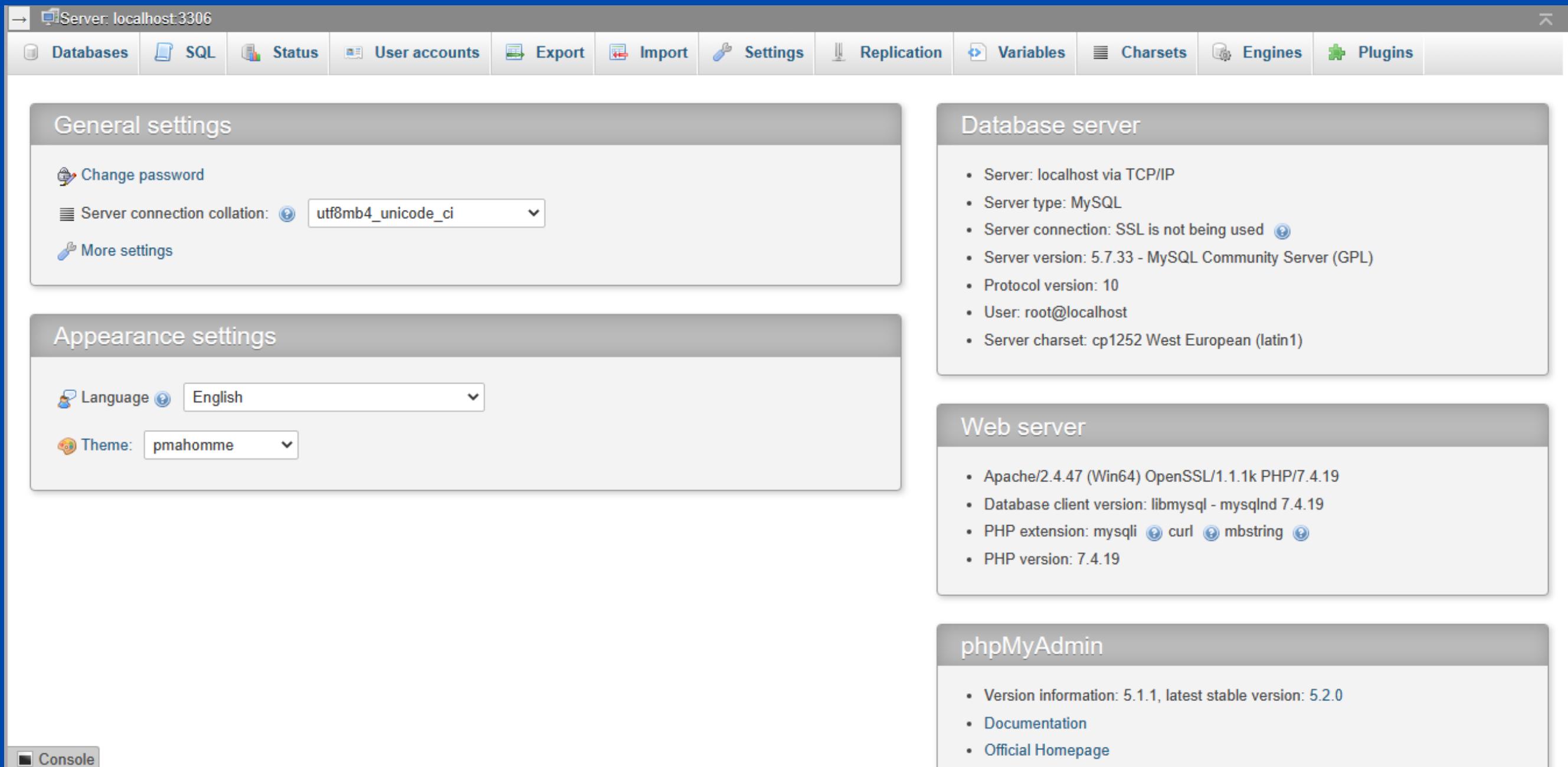
```
● ● ●  
1 private const DB_PASS = 'password';
```

Kesalahan saat mengisi konfigurasi untuk nama database / database tidak dikenal di database server.

```
SQLSTATE[HY000] [1049] Unknown database 'pdo'
```

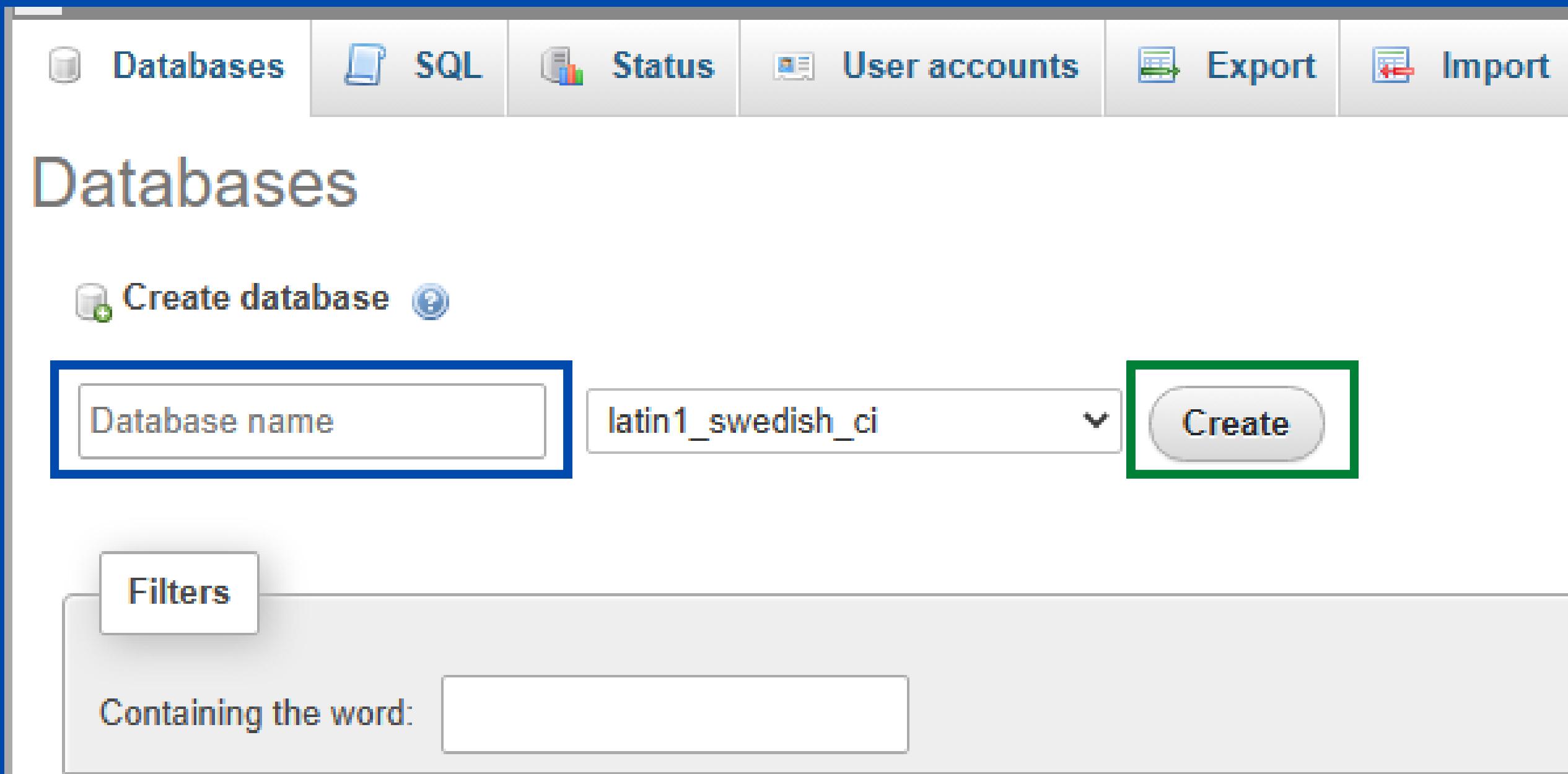
```
● ● ●  
1 private const DB_NAME = 'pdo';  
2
```

# Akses Database



Membuat Database Mysql menggunakan PHPMyAdmin, dengan cara mengakses URL "localhost/phpmyadmin".

# Membuat Database



Setelah berada di halaman PHPMyAdmin, kemudian pilih tab Database dan buatlah database sesuai dengan konfigurasi sebelumnya.

# Memilih Database

Databases

Create database [?](#)

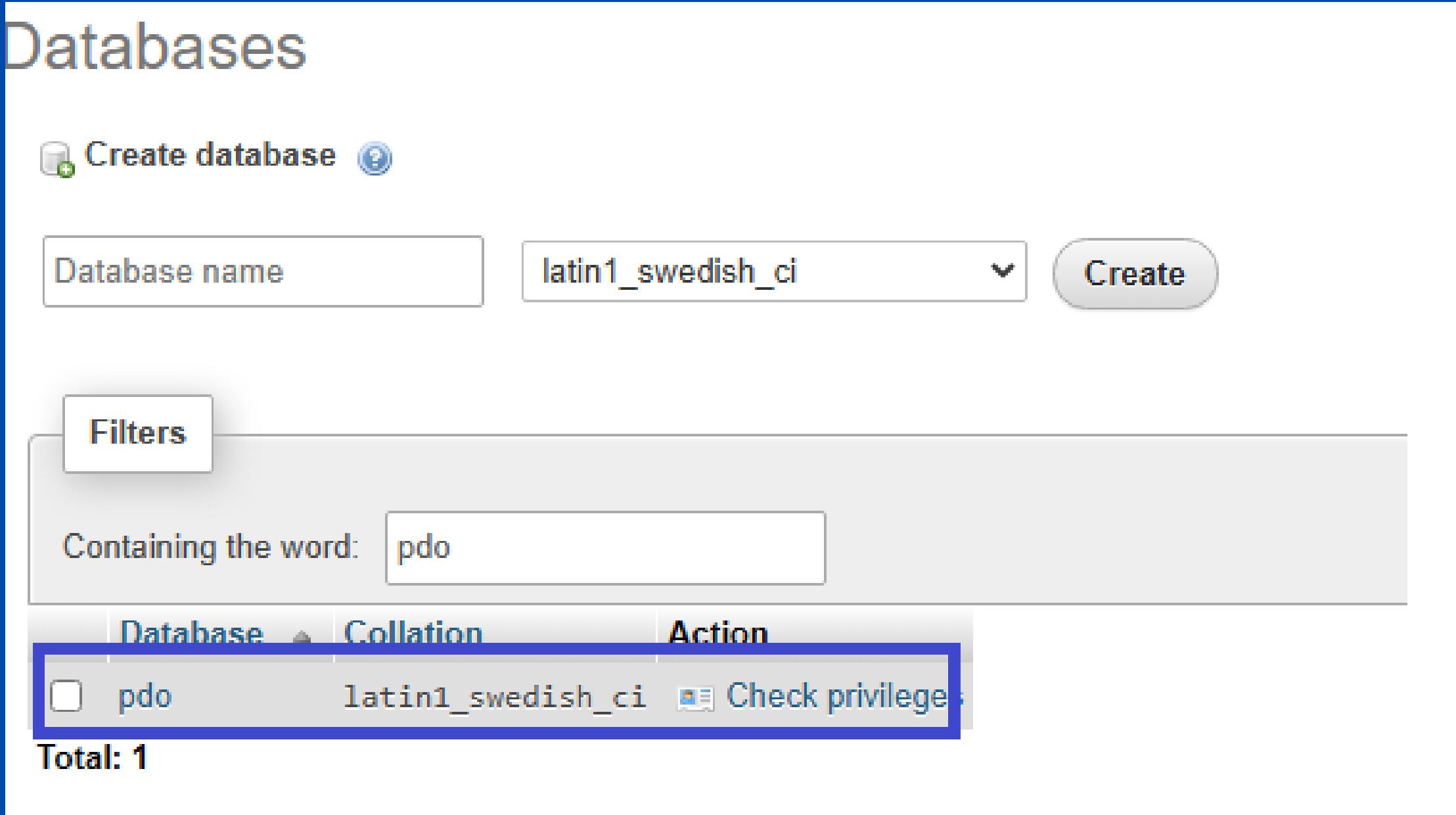
Database name: latin1\_swedish\_ci [▼](#) [Create](#)

[Filters](#)

Containing the word: pdo

Database	Collation	Action
<input type="checkbox"/> pdo	latin1_swedish_ci	<a href="#">Check privilege</a>

Total: 1



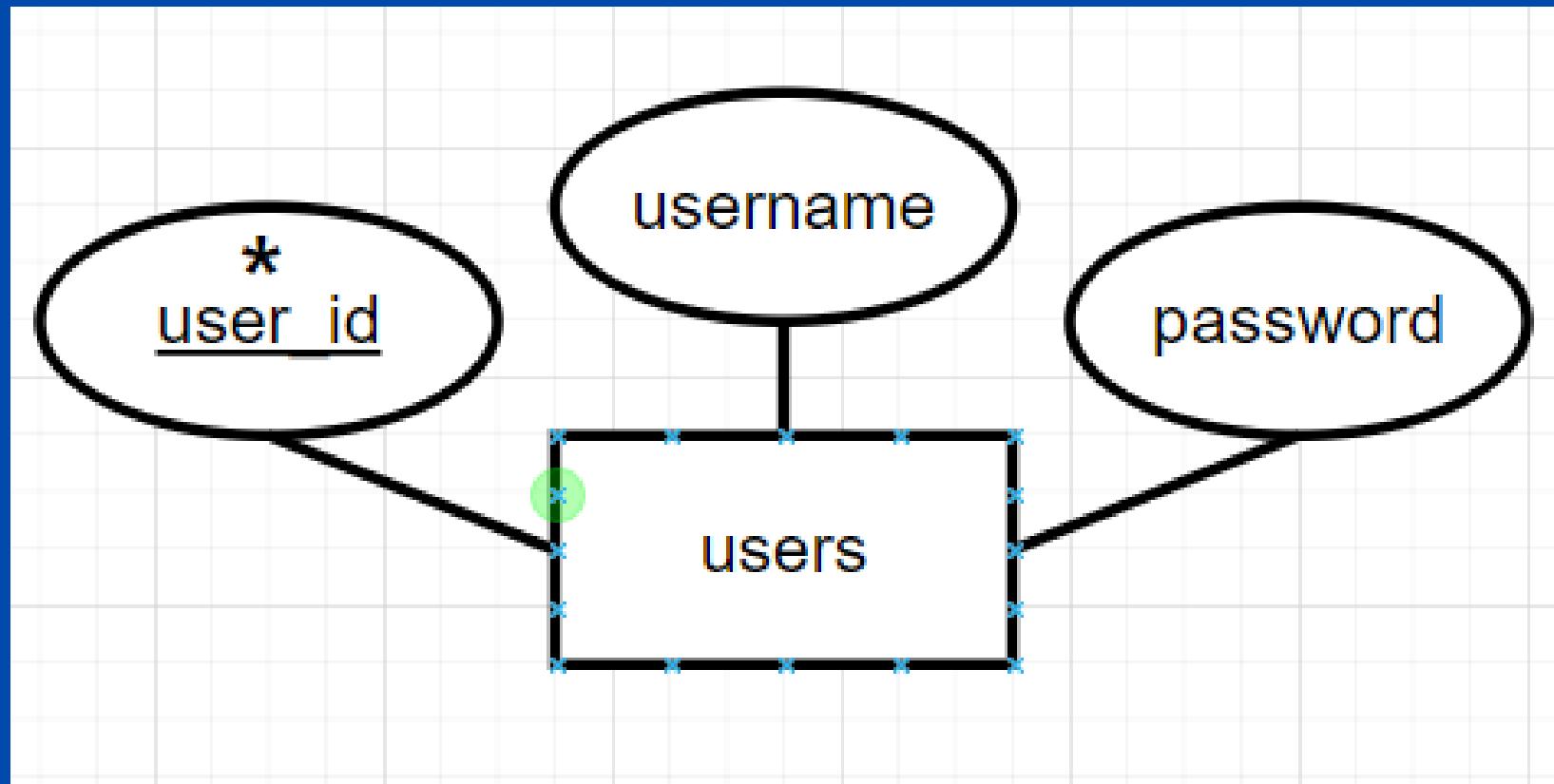
Selanjutnya pilih database yang sudah dibuat dan persiapan selanjutnya yaitu membuat table yang akan dikelola oleh PDO.

# Tampilan Halaman Database

The screenshot shows the MySQL Workbench application window. At the top, the server is set to 'localhost:3306' and the database is 'pdo'. Below the header, there is a navigation bar with several tabs: Structure, SQL (which is highlighted with a green border), Search, Query, Export, Import, Operations, Privileges, Routines, Events, and Triggers. A message in the main area states 'No tables found in database.' Below this, there is a 'Create table' button with a plus sign icon. Underneath it, there are two input fields: 'Name:' and 'Number of columns:', both containing the value '4'. At the bottom left is a 'Go' button.

Untuk membuat table yang ingin dirancang, pilih tab **SQL** untuk membuat table dengan perintah SQL (Structure Query Language)

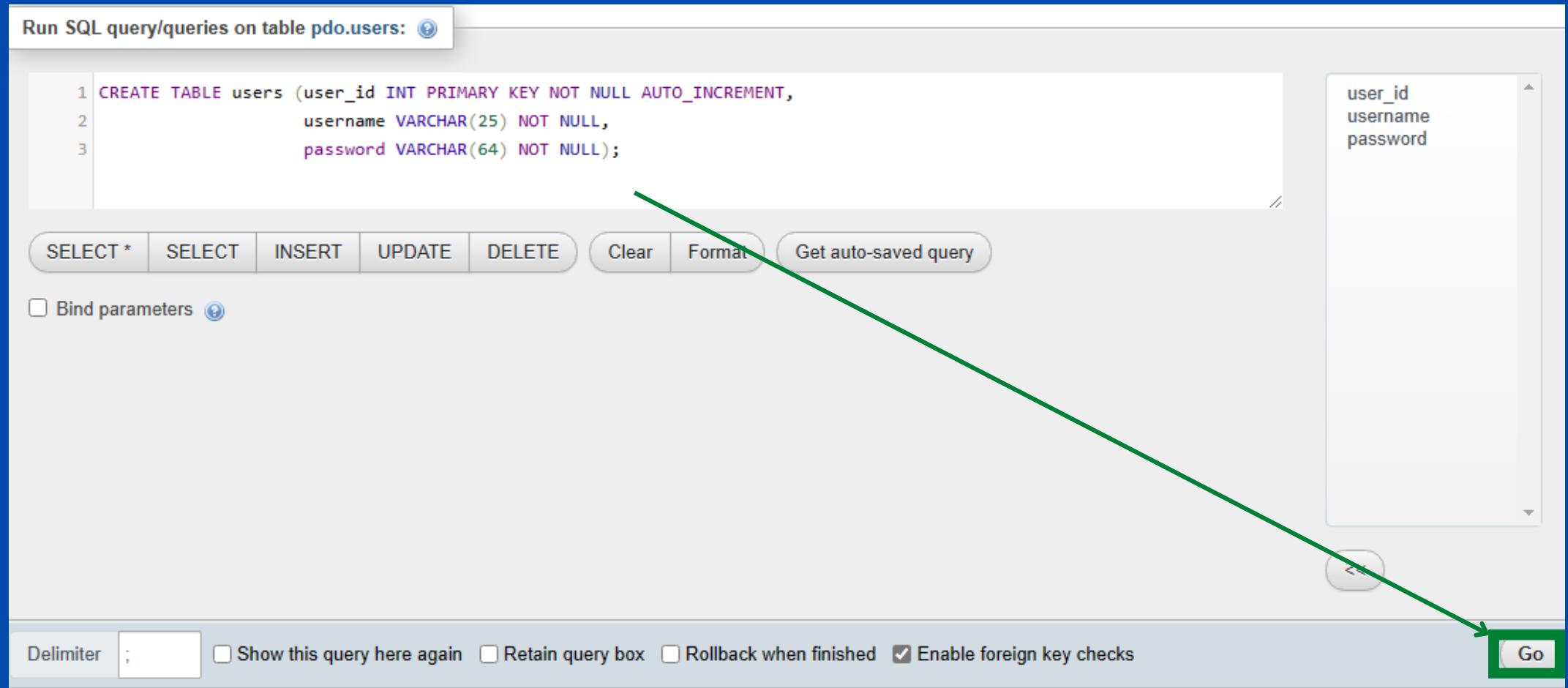
# Perancangan Table



No	Field	Type	Size	Index
1	user_id	int	11	Primary Key
2	username	varchar	25	
3	password	varchar	64	

Table yang akan dirancang adalah table **users** yang biasanya digunakan untuk melakukan autentikasi seperti login dan logout

# Pembuatan Table



This part of the image shows the same SQL code in a dark-themed environment. It includes three colored circles (red, yellow, green) at the top left. The code is identical to the one in the previous screenshot:

```
1 CREATE TABLE
2     users (
3         user_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
4         username VARCHAR(25) NOT NULL,
5         password VARCHAR(64) NOT NULL
6 );
```

Kemudian untuk membuat table users menggunakan perintah SQL seperti di gambar ini.

# Struktur Data Table

The screenshot shows the phpMyAdmin interface for a MySQL database. The top navigation bar indicates the connection is to 'Server: localhost3306', the database is 'pdo', and the current table is 'users'. Below the navigation bar is a toolbar with various management options: Browse, Structure (which is selected), SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers. Under the 'Structure' tab, there are two sub-options: 'Table structure' (selected) and 'Relation view'. The main content area displays the structure of the 'users' table. The table has three columns: 'user\_id' (int(11)), 'username' (varchar(25)), and 'password' (varchar(64)). The 'user\_id' column is defined as AUTO\_INCREMENT, primary key, and unique. The 'username' and 'password' columns have collation set to latin1\_swedish\_ci. The table structure is presented in a grid with columns for #, Name, Type, Collation, Attributes, Null, Default, Comments, Extra, and Action. At the bottom of the table structure grid, there are buttons for 'Check all', 'With selected:', and various actions like 'Browse', 'Change', 'Drop', 'Primary', 'Unique', 'Index', 'Spatial', and 'Fulltext'.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(11)			No	None		AUTO_INCREMENT	Change  Drop  More
2	username	varchar(25)	latin1_swedish_ci		No	None			Change  Drop  More
3	password	varchar(64)	latin1_swedish_ci		No	None			Change  Drop  More

Hasil struktur data pada table ketika sudah berhasil membuat sebuah table **users**.

# Sample Rows Data

```
Run SQL query/queries on table pdo.users: ⓘ  
1 INSERT INTO users (user_id, username, password)  
2 VALUES ('1','admin',md5('admin')),('2','operator',md5('operator'));
```



```
1 INSERT INTO users (user_id, username, password)  
2 VALUES ('1', 'admin', md5('admin'), '2', 'operator', md5('operator'));
```

Setelah membuat table **users**, untuk menampilkan baris data memerlukan sample dengan perintah **Insert** pada SQL

# Rows Data

The screenshot shows the phpMyAdmin interface for a MySQL database. The top navigation bar indicates the connection is to 'Server: localhost3306', the database is 'pdo', and the current table is 'users'. Below the navigation bar is a toolbar with links for 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', and 'Operations'. A success message in a yellow bar states: 'Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)'. The SQL query executed is 'SELECT \* FROM `users`'. There is a checkbox for 'Profiling' and links for 'Edit inline', 'Edit', 'Explain SQL', 'Create PHP code', and 'Refresh'. Below these controls are filters for 'Number of rows' (set to 25), 'Filter rows' (text input field 'Search this table'), and 'Sort by key' (dropdown menu set to 'None'). The main content area displays the 'users' table with two rows:

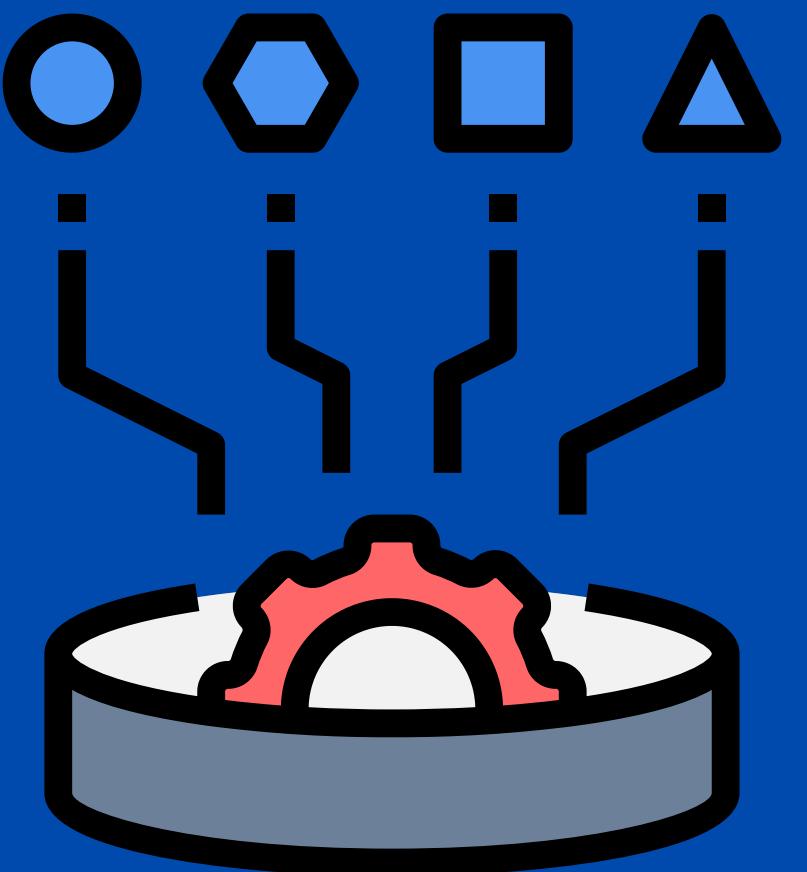
	user_id	username	password
<input type="checkbox"/>	1	admin	21232f297a57a5a743894a0e4a801fc3
<input type="checkbox"/>	2	operator	4b583376b2767b923c3e1da60d10de59

At the bottom of the table area are buttons for 'Check all', 'With selected:', 'Edit', 'Copy', 'Delete', and 'Export'.

Hasil baris data (rows) yang telah diinput melalui perintah SQL insert.

# API - CRUD (Create Read Update Delete)

CRUD atau Create Read Update & Delete suatu proses manipulasi data seperti menampilkan, menambahkan, mengubah dan menghapus data melalui sistem yang telah dirancang.



# Testing Tools API



**Postman**

Link Download

<https://www.postman.com/downloads/>

# Tampilan Awal Postman

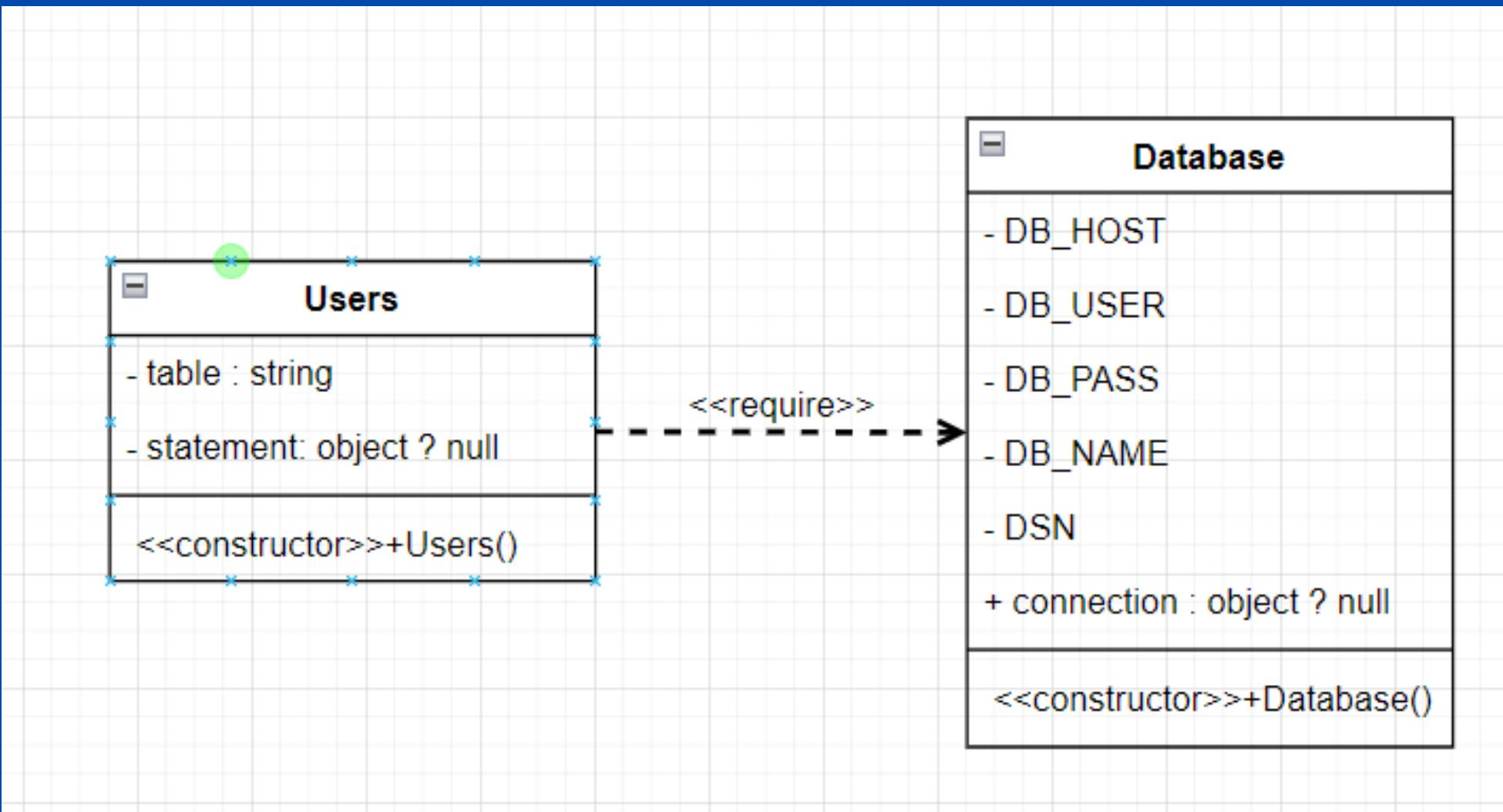
The screenshot shows the main interface of the Postman application. At the top, there is a navigation bar with links for Home, Workspaces, API Network, and Explore. A search bar labeled "Search Postman" is located on the right of the navigation bar. To the right of the search bar are icons for "Invite", "Settings", "Notifications", and "Upgrade". Below the navigation bar, the title "My Workspace" is displayed, along with "New" and "Import" buttons. On the far right, it says "No Environment".

The left sidebar contains a "Collections" section with a plus icon to add new collections. Other sections include APIs, Environments, Mock Servers, Monitors, Flows, and History. A large central area displays a collection named "My first collection" which contains two folders: "First folder inside collection" and "Second folder inside collection". Each folder has several requests listed under it.

A prominent call-to-action section in the center says "Create a collection for your requests" with a description: "A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it." It includes a "Create Collection" button and a "Open Workspace Overview" button.

At the bottom, there is a "Create a new request:" section with icons for GET, POST, PUT, and PATCH. The bottom navigation bar includes links for Online, Find and Replace, Console, Cookies, Capture requests, Runner, Trash, and Help.

# Perancangan Class Diagram Implementasi CRUD

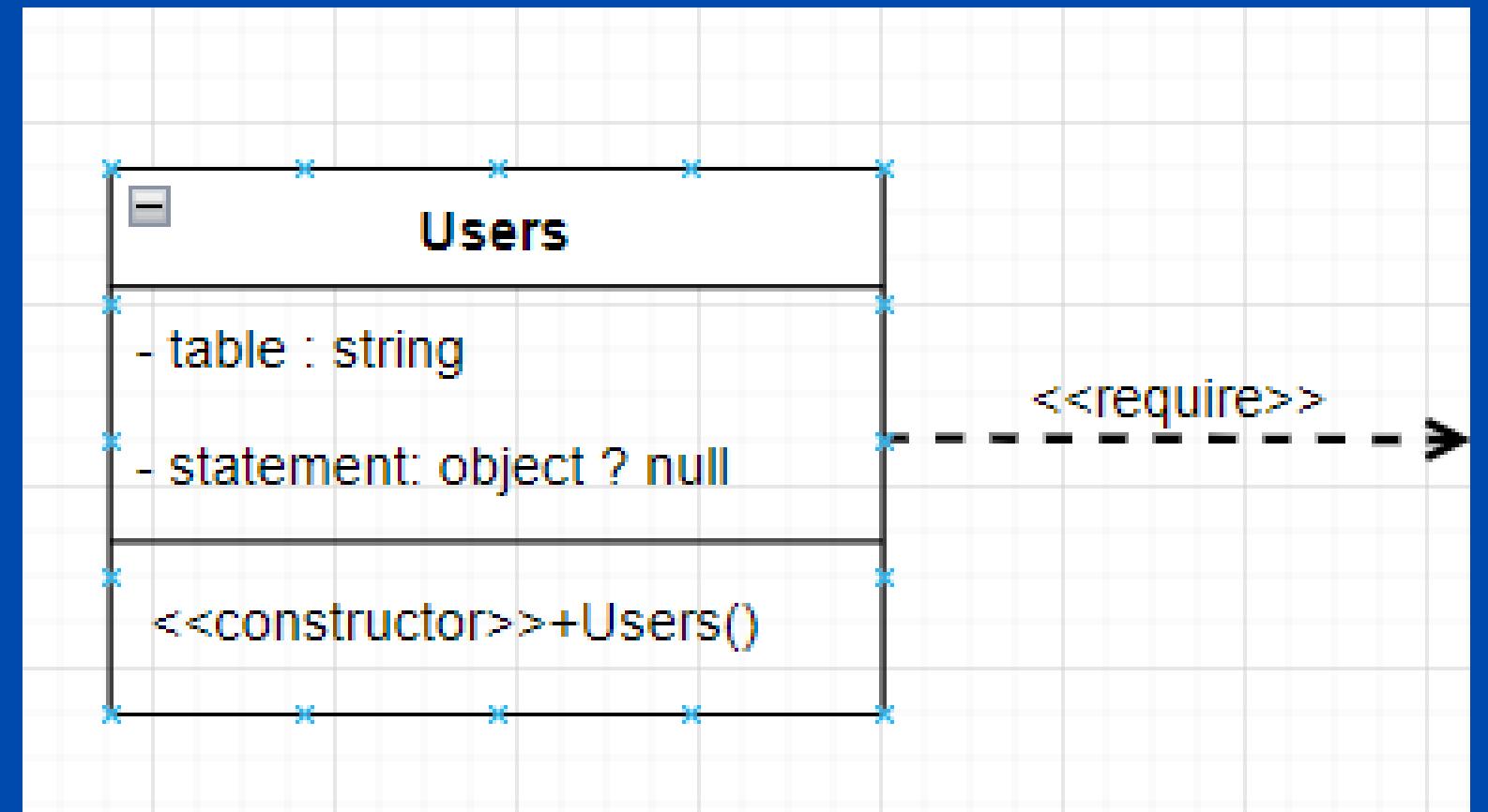


Untuk implementasi crud membutuhkan class **Users** yang dimana class **Database** akan dibutuhkan di class nya.

# Implementasi CRUD

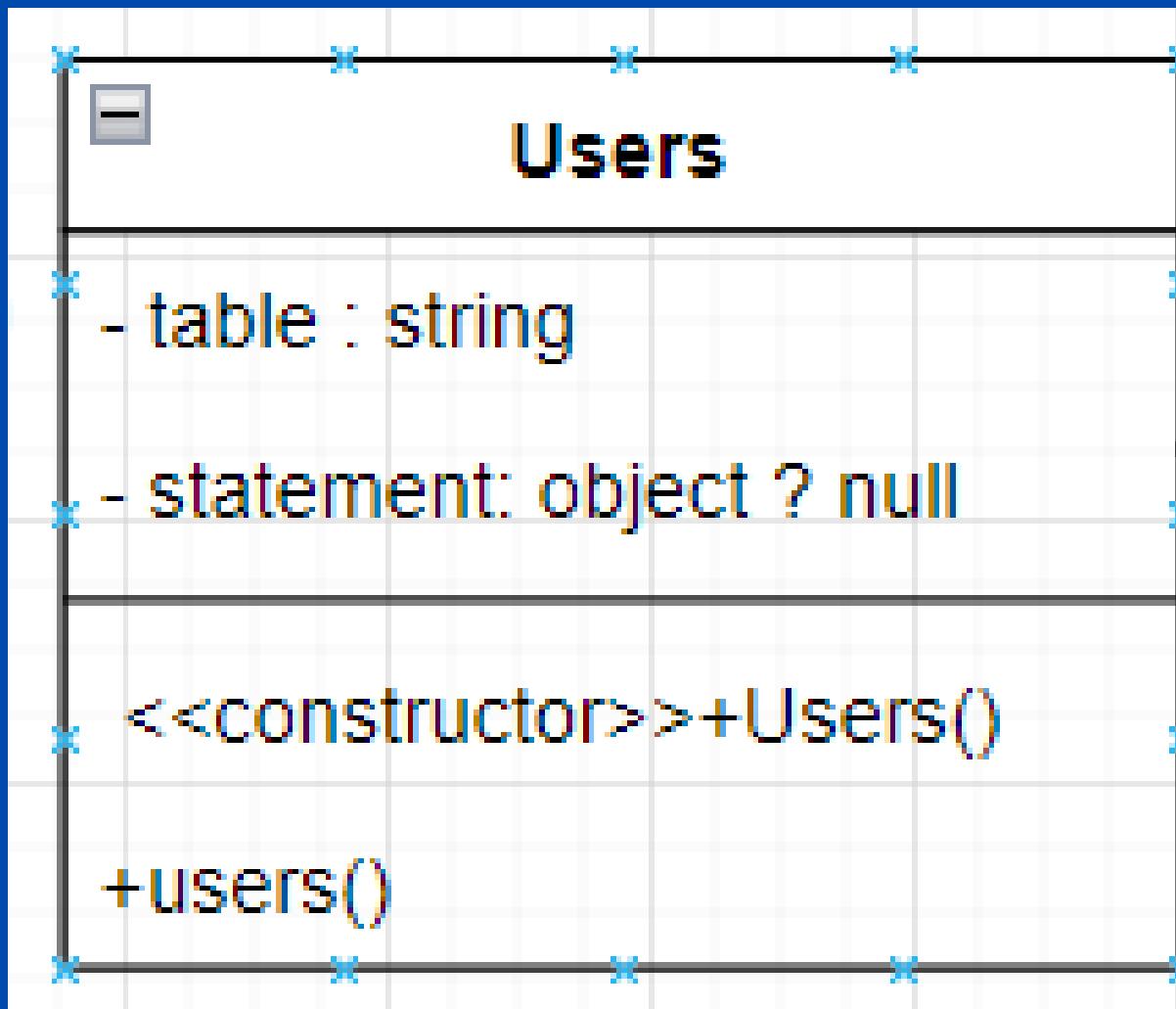


```
1 <?php
2
3 require_once('Database.php');
4
5 class Users
6 {
7     private string $table = 'users';
8     private ?object $statement;
9
10    public function __construct()
11    {
12        $this->statement = new Database();
13        $this->statement = $this->statement->connection;
14    }
15 }
```



Buatlah file dengan nama **Users.php** kemudian untuk memanggil file **Database.php** dengan keyword **require\_once** atau istilah di class diagramnya adalah dependency, dan instance object **Database** di dalam method **construct** untuk menggunakan property **connection** yang ada di dalam class **Database**.

# Menampilkan List Data (Read)



```
● ● ●
1 <?php
2
3 require_once('Database.php');
4
5 class Users
6 {
7     private string $table = 'users';
8     private ?object $statement;
9
10    public function __construct()
11    {
12        $this->statement = new Database();
13        $this->statement = $this->statement->connection;
14    }
15
16    public function users()
17    {
18    }
19}
20
```

Untuk menampilkan list data dengan menambahkan method users

# Membuat Query PDO

```
1 public function users()
2 {
3     $query = "SELECT * FROM users";
4 }
```

Setelah membuat fungsi **users** untuk membuat query, dengan perintah SELECT dan ditampung dalam sebuah variable.

# Menggunakan Property Select Table

```
1  <?php
2
3  require_once('Database.php');
4
5  class Users
6  {
7      private string $table = 'users';
8      private ?object $statement;
9
10     public function __construct()
11     {
12         $this->statement = new Database();
13         $this->statement = $this->statement->connection;
14     }
15
16     public function users()
17     {
18         $query = "SELECT * FROM $this->table";
19     }
20 }
```

Menggunakan property **table** karena sudah disediakan pada sebelumnya.

# Prepare Statement Query



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5 }
```

Setelah menentukan query SELECT, kemudian query akan di tampung dalam method prepare dengan argument berbentuk variable **query**. Tujuannya supaya query yang telah dibuat akan di tumpung terlebih dahulu sebelum dieksekusi query tersebut.

# Return Prepare Statement & Debugging



```
1  <?php
2
3  require_once('Database.php');
4
5  class Users
6  {
7      private string $table = 'users';
8      private ?object $statement;
9
10     public function __construct()
11     {
12         $this->statement = new Database();
13         $this->statement = $this->statement->connection;
14     }
15
16     public function users()
17     {
18         $query = "SELECT * FROM $this->table";
19         $statement = $this->statement->prepare($query);
20         return $statement;
21     }
22 }
23
24
25 $users = new Users();
26 $users = $users->users();
27 var_dump($users);
```

```
object(PDOStatement)#4 (1) { ["queryString"]=> string(19) "SELECT * FROM users" }
```

Selanjutnya statement yang sudah dibuat akan di return di dalam method **users** dan membuat object dari class **Users**, serta memanggil method users untuk melihat hasil dari statement tersebut dan memastikan query yang digunakan sesuai dengan table yang dirancang.

# Execute Statement

```
● ● ●  
1 public function users()  
2 {  
3     $query = "SELECT * FROM $this→table";  
4     $statement = $this→statement→prepare($query);  
5     $statement→execute();  
6     return $statement;  
7 }
```

Setelah melakukan pengecekan prepare statement, dengan menambahkan method execute, untuk menjalankan query ke database server.

# Debugging Execute Statement

```
● ● ●  
1 <?php  
2 require_once('Database.php');  
3  
4 class Users  
5 {  
6     private string $table = 'users';  
7     private ?object $statement;  
8  
9     public function __construct()  
10    {  
11        $this->statement = new Database();  
12        $this->statement = $this->statement->connection;  
13    }  
14  
15    public function users()  
16    {  
17        $query = "SELECT * FROM $this->table";  
18        $statement = $this->statement->prepare($query);  
19        $statement->execute();  
20        var_dump($statement->execute()); die;  
21        return $statement;  
22    }  
23 }  
24  
25  
26  
27 $users = new Users();  
28 $users = $users->users();  
29 var_dump($users);
```

Benar

bool(true)

Error

```
Fatal error: Uncaught PDOException: SQLSTATE[42S02]: Base table or view not found: 1146 Table 'pdo.user' doesn't exist in C:\laragon\www\mardirabootcamp\PDO\Users.php(20): PDOStatement->execute() #1 C:\laragon\www\mardirabootcamp\PDO\Users.php on line 20
```

Debugging untuk melihat hasil eksekusi query yang dilakukan jika tidak ada kendala dalam query maka akan menampilkan output boolean true, sebaliknya akan muncul pesan error seperti gambar di atas.

# Fetching Rows / Menangkap Baris Data



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);
7     return $statement;
8 }
```

Kemudian untuk menangkap baris data yang telah dieksekusi menambahkan object yang memiliki method fetchAll dengan argument static constant yang dimiliki oleh class PDO yaitu FETCH\_OBJ.

# Debugging Fetch Rows



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);
7     echo "<pre>";
8     var_dump($users); die;
9     echo "</pre>";
10    return $statement;
11 }
```

```
array(2) {
[0]=>
object(stdClass)#6 (3) {
    ["user_id"]=>
        string(1) "1"
    ["username"]=>
        string(5) "admin"
    ["password"]=>
        string(32) "ad8c0863e80c639ed29473efb54ea4fc"
}
[1]=>
object(stdClass)#7 (3) {
    ["user_id"]=>
        string(1) "4"
    ["username"]=>
        string(8) "operator"
    ["password"]=>
        string(32) "4b583376b2767b923c3e1da60d10de59"
}
}
```

Kemudian untuk menangkap baris data yang telah dieksekusi menambahkan object yang memiliki method fetchAll dengan argument static constant yang dimiliki oleh class PDO yaitu FETCH\_OBJ.

# Jenis Fetch Rows

- **Methods**
  - `fetchAll(parameter)` = Menangkap semua baris data
  - `fetch(parameter)` = Menangkap satu baris data
- **Static Constant Arguments**
  - `PDO::FETCH_NUM` = Array Terindeks / Indexed Array
  - `PDO::FETCH_ASSOC` = Array Asosiatif / Associative Array
  - `PDO::FETCH_BOTH` = Array Gabungan (Terindeks & Asosiatif)
  - `PDO::FETCH_OBJ` = Object
  - `PDO::FETCH_LAZY` = Flexible (Tidak berlaku untuk `fetchAll`)

# PDO::FETCH\_NUM



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_NUM);
7     echo '<pre>';
8     print_r($users);
9     echo '</pre>';
10    return $statement;
11 }
```

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => admin
            [2] => ad8c0863e80c639ed29473efb54ea4fc
        )
    [1] => Array
        (
            [0] => 2
            [1] => operator
            [2] => 4b583376b2767b923c3e1da60d10de59
        )
)
```

**PDO::FETCH\_NUM** yaitu static constant argument pada method fetch berfungsi untuk mengembalikan baris data menjadi array yang sifatnya terindeks / index array.

# Implementasi PDO::FETCH\_NUM

```
● ● ●  
1 public function users()  
2 {  
3     $query = "SELECT * FROM $this->table";  
4     $statement = $this->statement->prepare($query);  
5     $statement->execute();  
6     $users = $statement->fetchAll(PDO::FETCH_NUM);  
7     foreach ($users as $user) {  
8         echo 'Username:' . $user[1] . '<br> Password:' . $user[2] . '<br>';  
9     }  
10    return $statement;  
11 }
```

Username:admin

Password: ad8c0863e80c639ed29473efb54ea4fc

Username:operator

Password: 4b583376b2767b923c3e1da60d10de59

# PDO::FETCH\_ASSOC



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_ASSOC);
7     echo "<pre>";
8     print_r($users);
9     echo "</pre>";
10    return $statement;
11 }
```

```
Array
(
    [0] => Array
    (
        [user_id] => 1
        [username] => admin
        [password] => ad8c0863e80c639ed29473efb54ea4fc
    )
    [1] => Array
    (
        [user_id] => 2
        [username] => operator
        [password] => 4b583376b2767b923c3e1da60d10de59
    )
)
```

**PDO::FETCH\_ASSOC** merupakan static constant argument pada method fetch bertujuan untuk mengembalikan baris data menjadi array yang sifatnya asosiatif/ associative array.

# Implementasi PDO::FETCH\_ASSOC



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_ASSOC);
7     foreach ($users as $user) {
8         echo 'Username:' . $user['username'] . '<br> Password:' . $user['password'] . '<br>';
9     }
10    return $statement;
11 }
```

Username:admin

Password: ad8c0863e80c639ed29473efb54ea4fc

Username:operator

Password: 4b583376b2767b923c3e1da60d10de59

# PDO::FETCH\_BOTH



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_BOTH);
7     echo "<pre>";
8     print_r($users);
9     echo "</pre>";
10    return $statement;
11 }
```

```
Array
(
    [0] => Array
    (
        [user_id] => 1
        [0] => 1
        [username] => admin
        [1] => admin
        [password] => ad8c0863e80c639ed29473efb54ea4fc
        [2] => ad8c0863e80c639ed29473efb54ea4fc
    )
    [1] => Array
    (
        [user_id] => 2
        [0] => 2
        [username] => operator
        [1] => operator
        [password] => 4b583376b2767b923c3e1da60d10de59
        [2] => 4b583376b2767b923c3e1da60d10de59
    )
)
```

PDO::FETCH\_BOTH adalah static constant argument pada method fetch untuk mengembalikan baris data menjadi gabungan dari array index dan array asosiatif

# Implementasi PDO::FETCH\_BOTH



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_BOTH);
7     foreach ($users as $user) {
8         echo 'Username:' . $user['username'] . '<br> Password:' . $user['password'] . '<br>';
9     }
10    return $statement;
11 }
```

atau



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_BOTH);
7     foreach ($users as $user) {
8         echo 'Username:' . $user[1] . '<br> Password:' . $user[2] . '<br>';
9     }
10    return $statement;
11 }
```

```
Username:admin
Password: ad8c0863e80c639ed29473efb54ea4fc
Username:operator
Password: 4b583376b2767b923c3e1da60d10de59
```

# PDO::FETCH\_OBJ



```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);
7     echo '<pre>';
8     print_r($users);
9     echo '</pre>';
10    return $statement;
11 }
```

```
Array
(
    [0] => stdClass Object
        (
            [user_id] => 1
            [username] => admin
            [password] => ad8c0863e80c639ed29473efb54ea4fc
        )

    [1] => stdClass Object
        (
            [user_id] => 2
            [username] => operator
            [password] => 4b583376b2767b923c3e1da60d10de59
        )
)
```

**PDO::FETCH\_OBJECT** adalah static constant argument pada method fetch bertujuan untuk mengembalikan baris data menjadi object.

# Implementasi PDO::FETCH\_OBJ

```
1 public function users()
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);
7
8     foreach ($users as $user) {
9         echo 'Username:' . $user->username . '<br> Password:' . $user->password . '<br>';
10    }
11    return $statement;
12 }
```

Username:admin

Password: ad8c0863e80c639ed29473efb54ea4fc

Username:operator

Password: 4b583376b2767b923c3e1da60d10de59

# API Untuk Menampilkan Data

```
● ● ●  
1 public function users() : void  
2 {  
3     $query = "SELECT * FROM $this->table";  
4     $statement = $this->statement->prepare($query);  
5     $statement->execute();  
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);  
7     $response = [  
8         'message' => 'Data User Berhasil Ditampilkan',  
9         'data' => $users  
10    ];  
11    echo json_encode($response);  
12 }
```

Untuk membuat response untuk API menampilkan data yaitu dengan menambahkan response berupa array dengan key dan value yang ingin dikonversi menjadi JSON

# Implementasi API Fetch Data

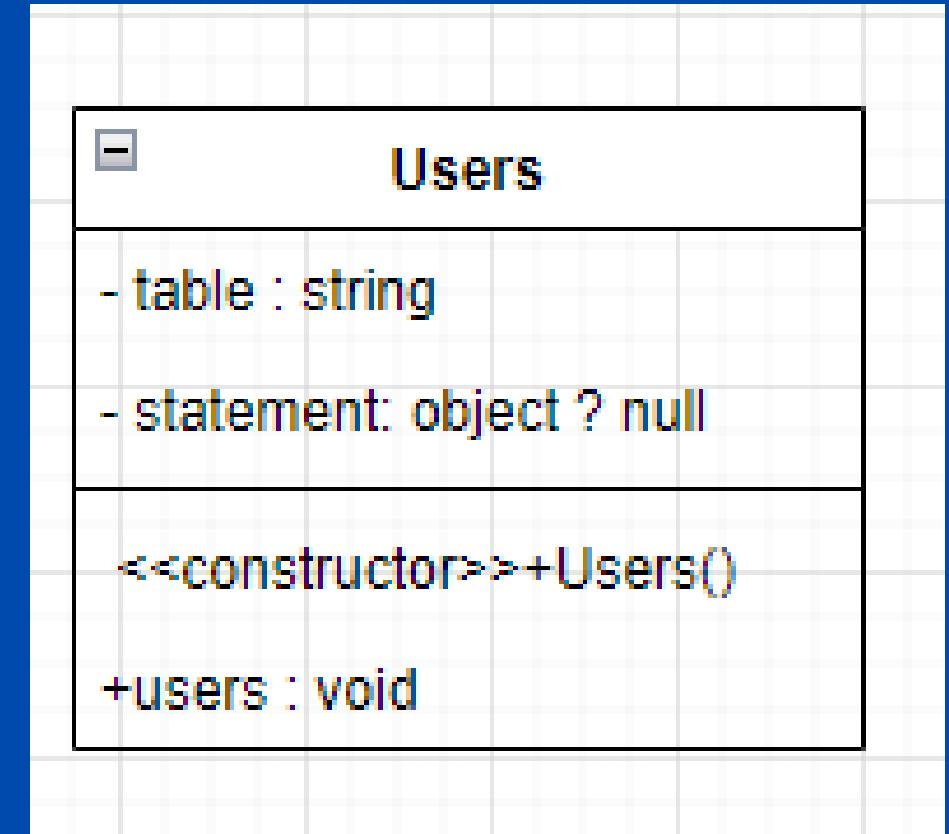
●

●

●

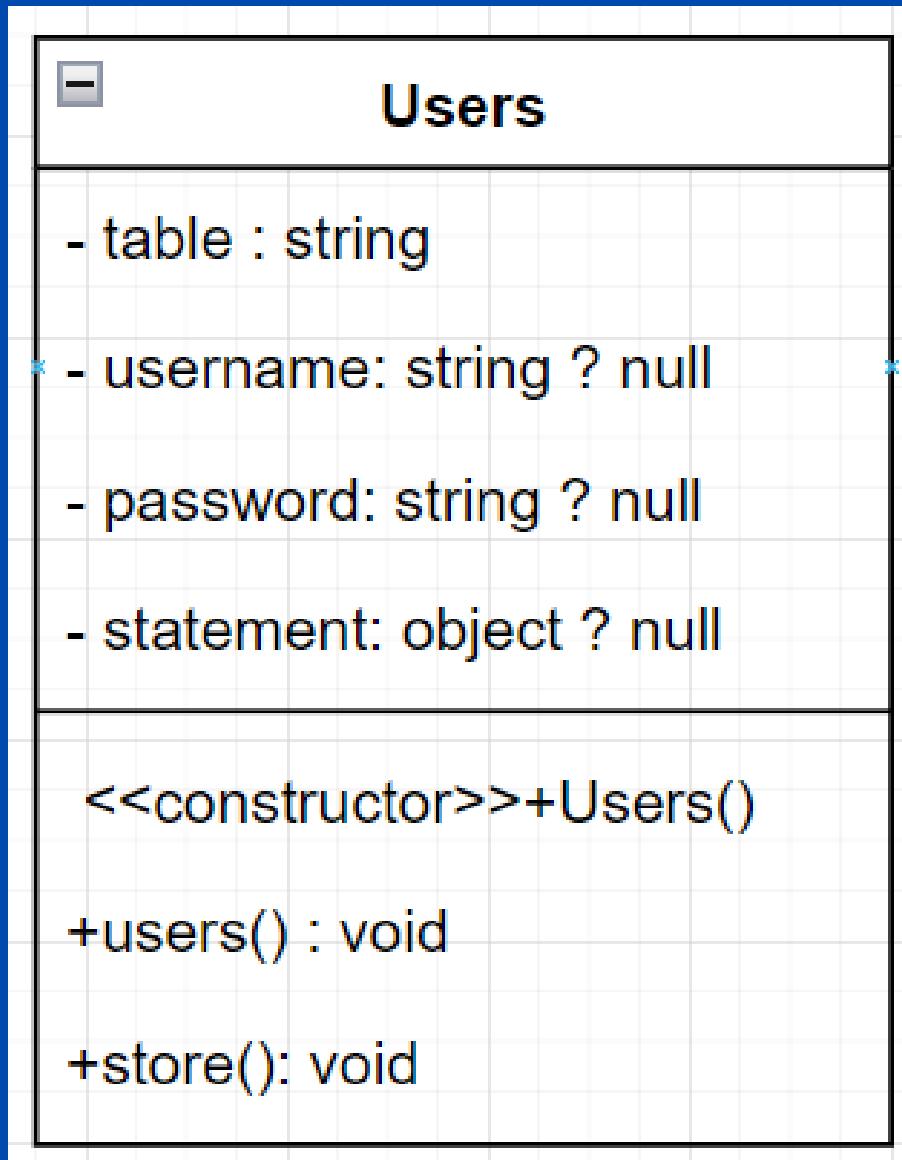
```
1 <?php
2 require_once('Database.php');
3
4 class Users
5 {
6     private string $table = 'users';
7     private ?object $statement;
8
9     public function __construct()
10    {
11        $this->statement = new Database();
12        $this->statement = $this->statement->connection;
13    }
14
15    public function users() : void
16    {
17        $query = "SELECT * FROM $this->table";
18        $statement = $this->statement->prepare($query);
19        $statement->execute();
20        $users = $statement->fetchAll(PDO::FETCH_OBJ);
21        $response = [
22            'message' => 'Data User Berhasil Ditampilkan',
23            'data' => $users
24        ];
25        echo json_encode($response);
26    }
27 }
28
29
30
31 $users = new Users();
32 $users->users();
33
34
```

```
{
    "message": "Data User Berhasil Ditampilkan",
    "data": [
        {
            "user_id": "1",
            "username": "admin",
            "password": "ad8c0863e80c639ed29473efb54ea4fc"
        },
        {
            "user_id": "2",
            "username": "operator",
            "password": "4b583376b2767b923c3e1da60d10de59"
        }
    ]
}
```



Secara tidak langsung kita telah membuat API untuk menampilkan baris data yang akan diberikan kepada Front-End Developer untuk diintegrasikan sesuai kebutuhan.

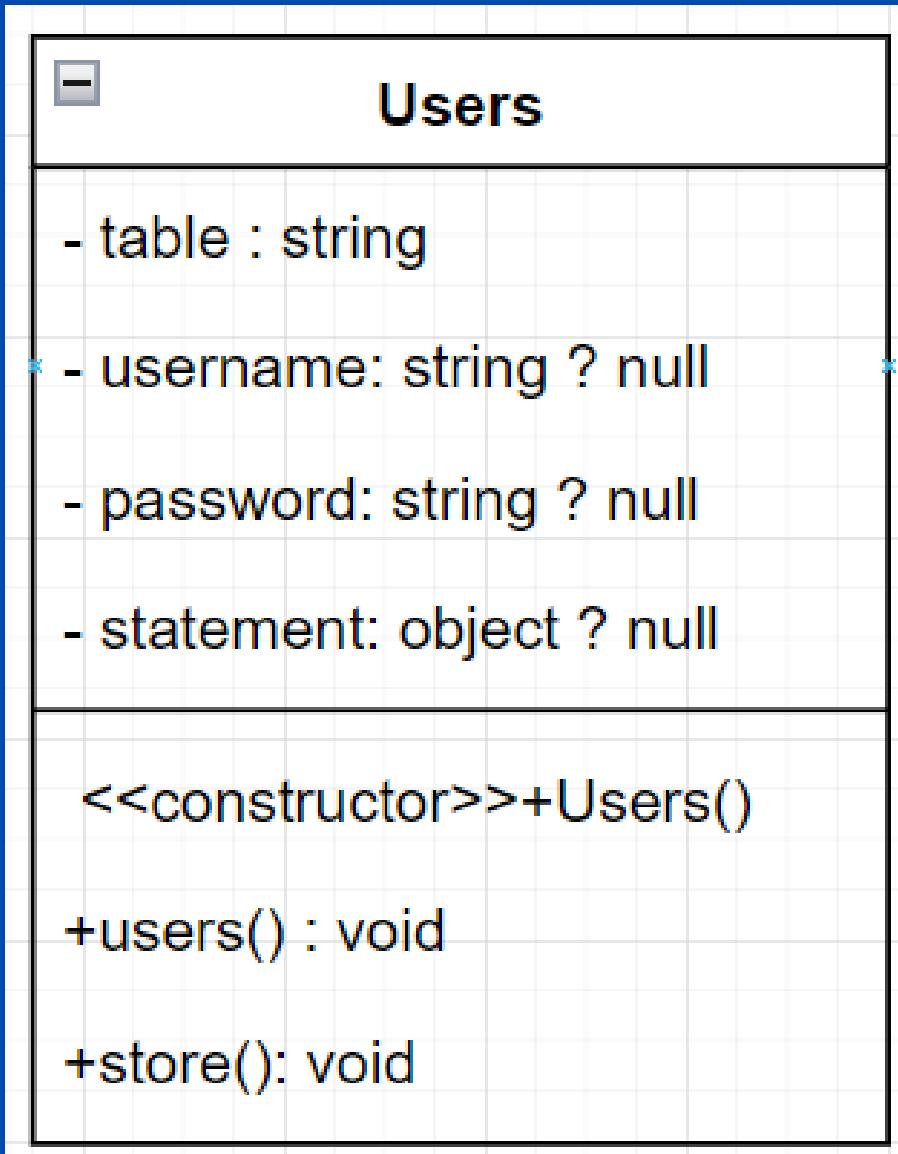
# Store / Insert Data (Create)



```
● ● ●
1 public function users(): void
2 {
3     $query = "SELECT * FROM $this->table";
4     $statement = $this->statement->prepare($query);
5     $statement->execute();
6     $users = $statement->fetchAll(PDO::FETCH_OBJ);
7     $response = [
8         'message' => 'Data User Berhasil Ditampilkan',
9         'data' => $users
10    ];
11    echo json_encode($response);
12 }
13
14 public function store(): void
15 {
16 }
```

Selanjutnya untuk menambahkan atau insert data, tambahkan methodnya terlebih dahulu dengan nama **store**

# Properties Untuk Field Data



```
1 <?php
2
3 require_once('Database.php');
4
5 class Users
6 {
7     private string $table = 'users';
8     private ?string $username;
9     private ?string $password;
10    private ?object $statement;
11}
```

Lalu tambahkan private properties username dan password yang berbentuk nullable types string seperti di atas.

# Menggunakan Field Pada Constructor

```
● ● ●  
1 private string $table = 'users';  
2 private ?string $username;  
3 private ?string $password;  
4 private ?object $statement;  
5  
6 public function __construct()  
7 {  
8     $this->statement = new Database();  
9     $this->statement = $this->statement->connection;  
10    $this->username = $_POST['username'] ?? null;  
11    $this->password = $_POST['password'] ?? null;  
12 }
```

Setelah menambahkan properties diatas, gunakan properties pada method constructor untuk menampung input data dengan menggunakan predefined variable yaitu **\$\_POST** dan menggunakan null coalescing jika data yang di input ada atau tidak.

# Query dan Statement Prepare



```
1 public function store(): void
2 {
3     $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
4     $statement = $this->statement->prepare($query);
5 }
```

Untuk membuat perintah menambahkan data dengan keyword INSERT INTO dengan nama tabelnya, setelah itu sebutkan field/column yang akan di tambahkan dengan valuenya, tetapi di PDO tidak bisa secara langsung memasukan parameter values dari properties, karena di PDO kita akan menggunakan **statement binding**.

# Statement Binding



```
1 public function store(): void
2 {
3     $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':username', $this->username);
6     $statement->bindParam(':password', $this->password);
7 }
```

```
3     $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)", →Binding Field
4     $statement->bindParam(':username', $this->username);
5     $statement->bindParam(':password', $this->password);
```

The diagram illustrates the mapping of a binding field to an argument field and its value. It shows the code snippet from line 3 to 5. The binding field ':username' and ':password' are highlighted with a black border. Arrows point from these fields to their corresponding argument fields '\$this->username' and '\$this->password' respectively, which are also highlighted with a black border. A third arrow points from the argument fields to the argument values '\$this->username' and '\$this->password'. Labels below the arrows identify them: 'Binding Field' for the top arrow, 'Argument Field' for the middle arrow, and 'Argument Value dari Property' for the bottom arrow.

Statement binding merupakan statement berupa method dari object yang disediakan oleh PDO untuk melakukan bind (mengaitkan) sebuah field pada table dengan 2 buah argument yaitu argument field dan argument value. Untuk menggunakan binding statement dengan field yang akan dikaitkan menggunakan simbol dan nama fieldnya ":field\_name".

# Eksekusi Query & Response API Insert

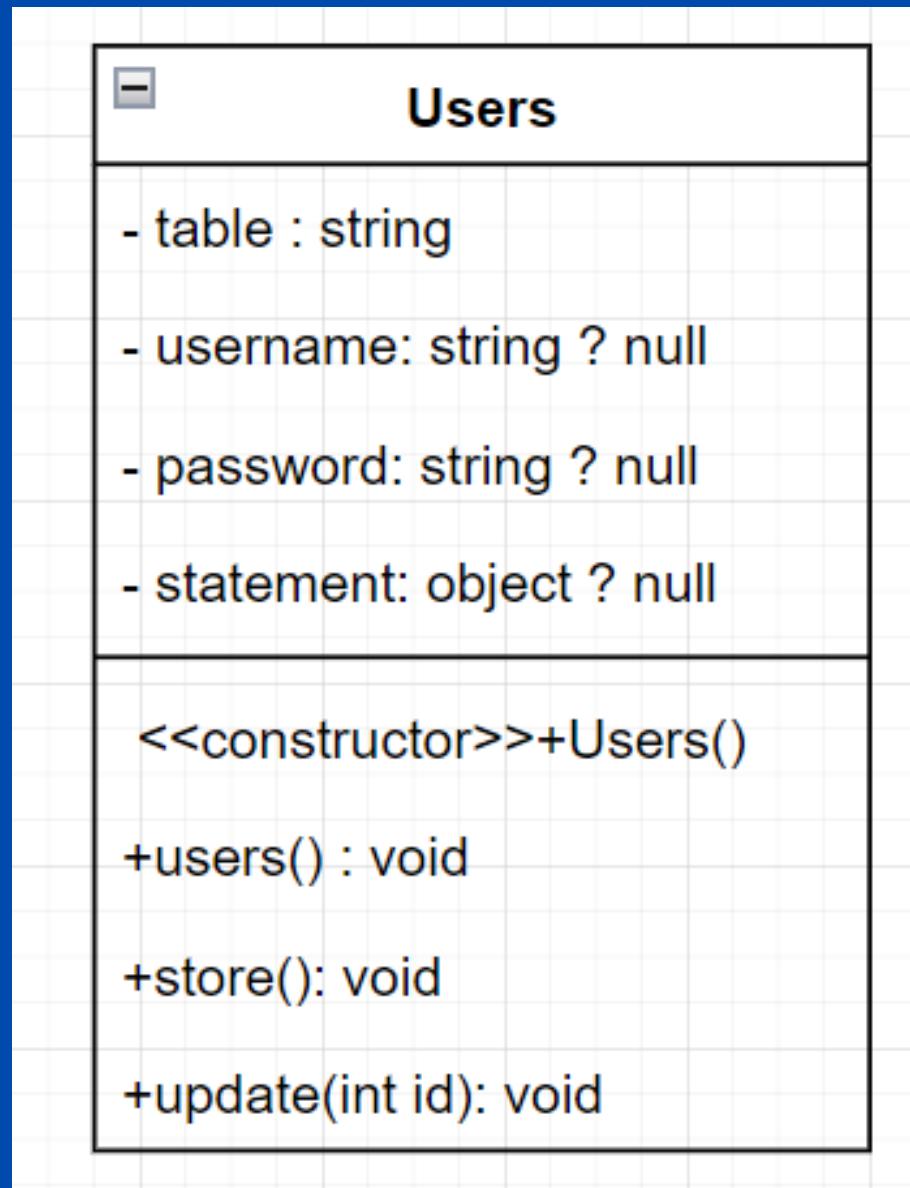
```
● ● ●  
1 public function insert(): void  
2 {  
3     $query = "INSERT INTO {$this→table} (username, password) VALUES (:username, :password)";  
4     $statement = $this→statement→prepare($query);  
5     $statement→bindParam(':username', $this→username);  
6     $statement→bindParam(':password', $this→password);  
7     $statement→execute();  
8     $response = [  
9         'message' => 'Data User Berhasil Ditambahkan',  
10    ];  
11    echo json_encode($response);  
12 }
```

Setelah melakukan statement binding, selanjutnya eksekusi query dan membuat response API seperti contoh di atas

# Implementasi Insert Data

```
1  <?php
2
3  require_once('Database.php');
4
5  class Users
6  {
7      private string $table = 'users';
8      private ?object $statement;
9
10     public function __construct()
11     {
12         $this->statement = new Database();
13         $this->statement = $this->statement->connection;
14         $this->username = $_POST['username'] ?? null;
15         $this->password = $_POST['password'] ?? null;
16     }
17
18     public function users(): void
19     {
20         $query = "SELECT * FROM $this->table";
21         $statement = $this->statement->prepare($query);
22         $statement->execute();
23         $users = $statement->fetchAll(PDO::FETCH_OBJ);
24         $response = [
25             'message' => 'Data User Berhasil Ditampilkan',
26             'data' => $users
27         ];
28         echo json_encode($response);
29     }
30
31     public function store(): void
32     {
33         $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
34         $statement = $this->statement->prepare($query);
35         $statement->bindParam(':username', $this->username);
36         $statement->bindParam(':password', $this->password);
37         $statement->execute();
38         $response = [
39             'message' => 'Data User Berhasil Dihapus',
40         ];
41         echo json_encode($response);
42     }
43 }
44
45 $users = new Users();
46 // $users->users();
47 $users->store();
48
```

# Update Data



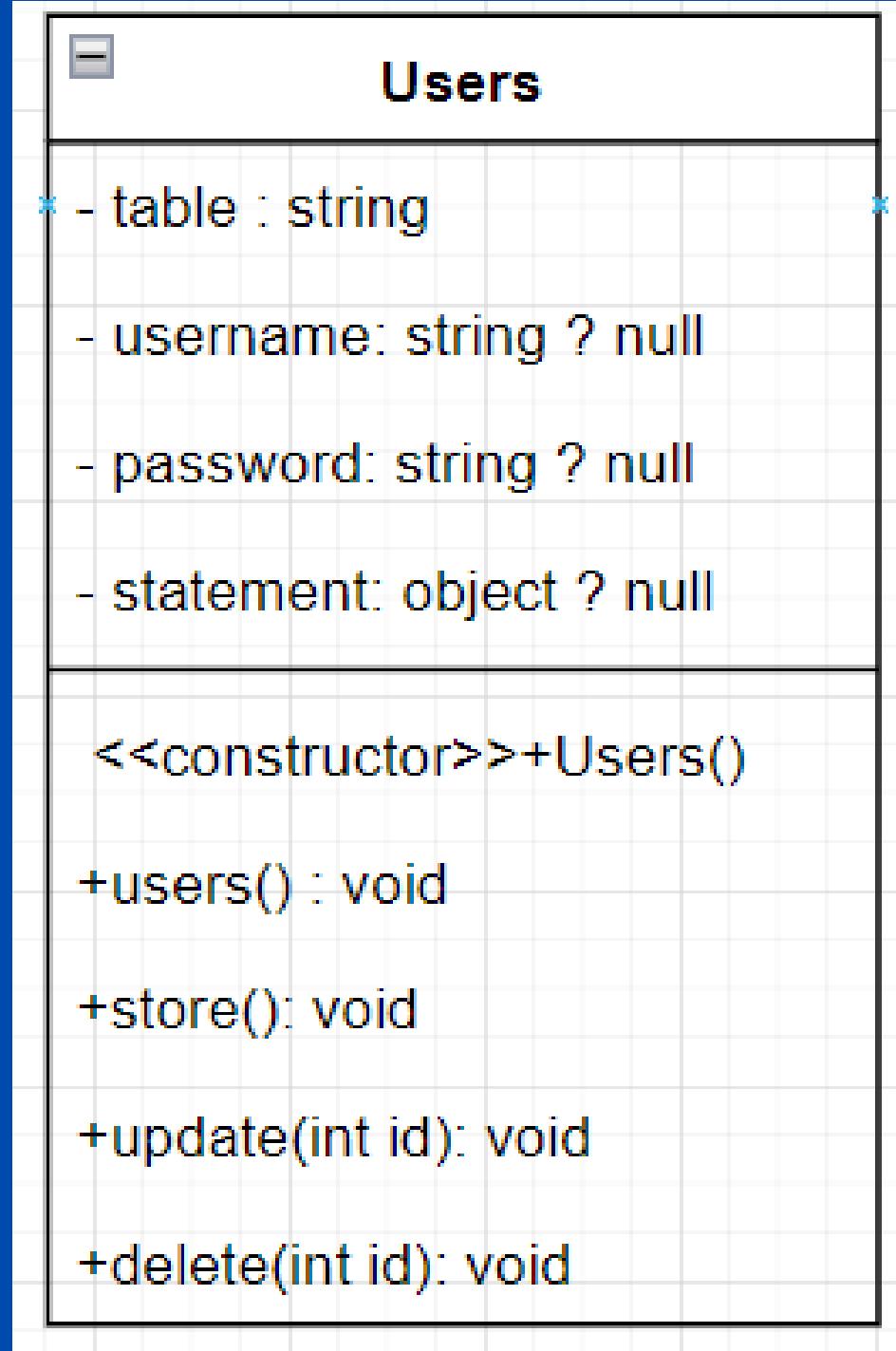
```
● ● ●
1 public function update(int $id): void
2 {
3     $query = "UPDATE {$this->table} SET username = :username, password = :password WHERE user_id = :user_id";
4     $statement = $this->statement->prepare($query);
5     $password = md5($this->password);
6     $statement->bindParam(':username', $this->username);
7     $statement->bindParam(':password', $password);
8     $statement->bindParam(':user_id', $id);
9     $statement->execute();
10    $response = [
11        'message' => 'Data User Berhasil Diubah',
12    ];
13    echo json_encode($response);
14 }
```

Selanjutnya akan membuat method untuk mengubah atau update data dengan membuat method dengan nama update, dan ditambah dengan parameter yang ditampung bertipe integer karena argument dari sebuah method yang di tampung adalah field id dari sebuah table.

# Implementasi Update Data

```
1  
2 $users = new Users();  
3 // $users→users();  
4 // $users→store();  
5 $users→update(1);
```

# Delete Data



```
● ● ●  
1 public function delete(int $id): void  
2 {  
3     $query = "DELETE FROM {$this->table} WHERE user_id = :user_id";  
4     $statement = $this->statement->prepare($query);  
5     $statement->bindParam(':user_id', $id);  
6     $statement->execute();  
7     $response = [  
8         'message' => 'Data User Berhasil Dihapus',  
9     ];  
10    echo json_encode($response);  
11 }
```

Method yang terakhir adalah menghapus atau delete data.

# Implementasi Delete Data

```
1 $users = new Users();
2 // $users->users();
3 // $users->store();
4 // $users->update(1);
5 $users->delete(2);
6
```

**Terimakasih**