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).

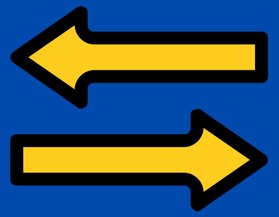
Konsep PDO

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 koneksinya 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

```php
<?php

$pdo = new PDO("mysql:host=localhost; dbname=database", 'username', 'password');
```

Mysqli Procedural

```php
<?php

$mysqli = mysqli_connect('localhost','username','password','database');
```

Mysqli OOP

```php
<?php

$mysqli = new mysqli('localhost','username','password','database');
```

Class Synopsis PDO

```php
<?php

class PDO {
/* Methods */
public __construct(
    string $dsn,
    ?string $username = null,
    ?string $password = null,
    ?array $options = null
)
public beginTransaction(): bool
public commit(): bool
public errorCode(): ?string
public errorInfo(): array
public exec(string $statement): int|false
public getAttribute(int $attribute): mixed
public static getAvailableDrivers(): array
public inTransaction(): bool
public lastInsertId(?string $name = null): string|false
public prepare(string $query, array $options = []): PDOStatement|false
public query(string $query, ?int $fetchMode = null): PDOStatement|false
public query(string $query, ?int $fetchMode = PDO::FETCH_COLUMN, int $colno): PDOStatement|false
public query(
    string $query,
    ?int $fetchMode = PDO::FETCH_CLASS,
    string $classname,
    array $constructorArgs
): PDOStatement|false
public query(string $query, ?int $fetchMode = PDO::FETCH_INTO, object $object): PDOStatement|false
public quote(string $string, int $type = PDO::PARAM_STR): string|false
public rollBack(): bool
public setAttribute(int $attribute, mixed $value): bool
}
```

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

Driver PDO

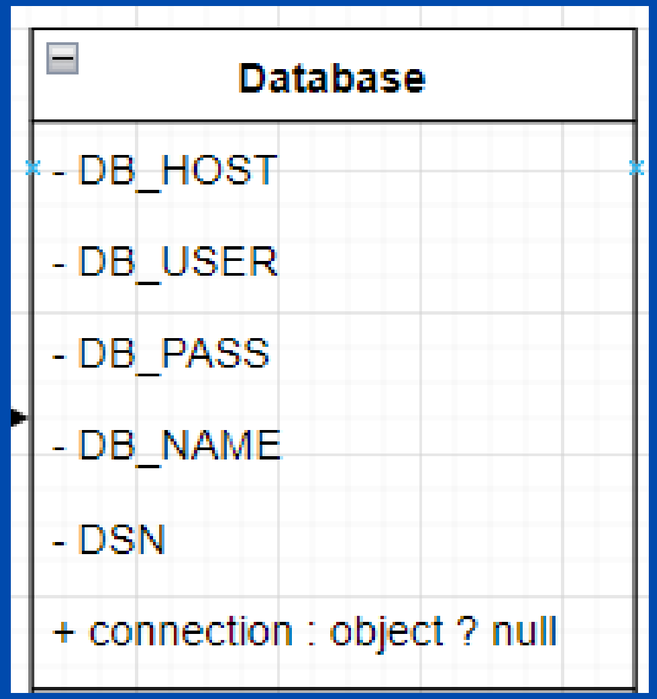
Untuk melihat driver database apa saja yang telah aktif dan bisa langsung digunakan, bisa menggunakan **static function** (Materi OOP), secara default driver yang bisa digunakan oleh PDO yaitu **Mysql**

```php
<?php

// function yang digunakan getAvailableDrivers adalah statis
// sehingga tidak perlu membuat objek baru
print_r(PDO::getAvailableDrivers());

```

```php
<?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:

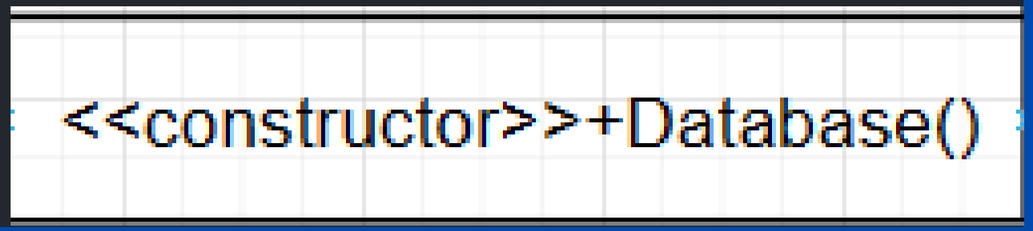
- Private constant DB_HOST (hostname)
- Private constant DB_USER (username mysql)
- Private DB_PASS (password mysql)
- Private constant DB_NAME (nama database yang akan dibuat)
- Private constant DSN (Data Source Name yang berisi dari driver, host dan nama database yang akan disatukan jadi satu source data
- Public variable connection yang digunakan untuk menghubungkan dari PDO ke database server

Properties Class Database Koneksi PDO

```php
<?php

class Database
{
    private const DB_HOST  = 'localhost';
    private const DB_USER = 'root';
    private const DB_PASS = '';
    private const DB_NAME = 'pdo';
    private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';
    public ?object $connection = null;
}
```

Constructor Koneksi PDO

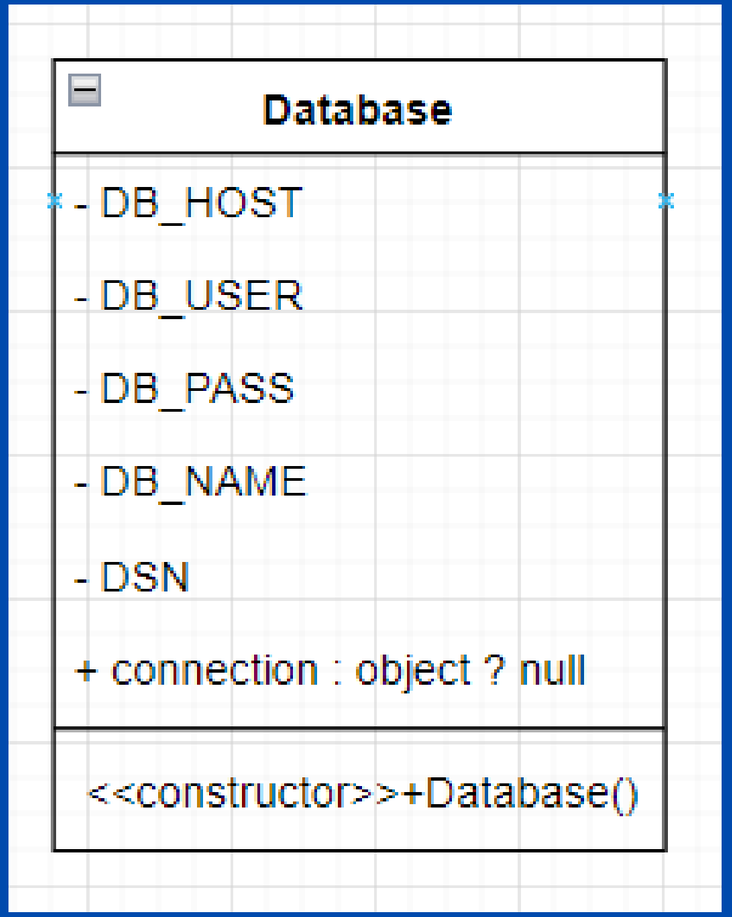
```php
public function __construct()
{
    try {
        $this->connection = new PDO(
            self::DSN,
            self::DB_USER,
            self::DB_PASS
        );
        $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
    return $this->connection;
}
```



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

```php
<?php

class Database
{
    private const DB_HOST  = 'localhost';
    private const DB_USER = 'root';
    private const DB_PASS = '';
    private const DB_NAME = 'pdo';
    private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';
    public ?object $connection = null;

    public function __construct()
    {
        try {
            $this->connection = new PDO(
                self::DSN,
                self::DB_USER,
                self::DB_PASS
            );
            $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch (PDOException $e) {
            echo $e->getMessage();
        }
        return $this->connection;
    }
}
```



Membuat Object Untuk Testing Koneksi

```php
<?php

class Database
{
    private const DB_HOST  = 'localhost';
    private const DB_USER = 'root';
    private const DB_PASS = '';
    private const DB_NAME = 'pdo';
    private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';
    public ?object $connection = null;

    public function __construct()
    {
        try {
            $this->connection = new PDO(
                self::DSN,
                self::DB_USER,
                self::DB_PASS
            );
            $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch (PDOException $e) {
            echo $e->getMessage();
        }
        return $this->connection;
    }
}

$database = new Database();

```

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

Testing Koneksi Database

```php
public function __construct()
{
    try {
        $this->connection = new PDO(
            self::DSN,
            self::DB_USER,
            self::DB_PASS
        );
        $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
    return $this->connection;
}
```

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.
```

```php
private const DB_HOST  = 'locahost';
```

Kesalahan saat mengisi konfigurasi untuk username ketika tidak menggunakan password

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

```php
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)
```

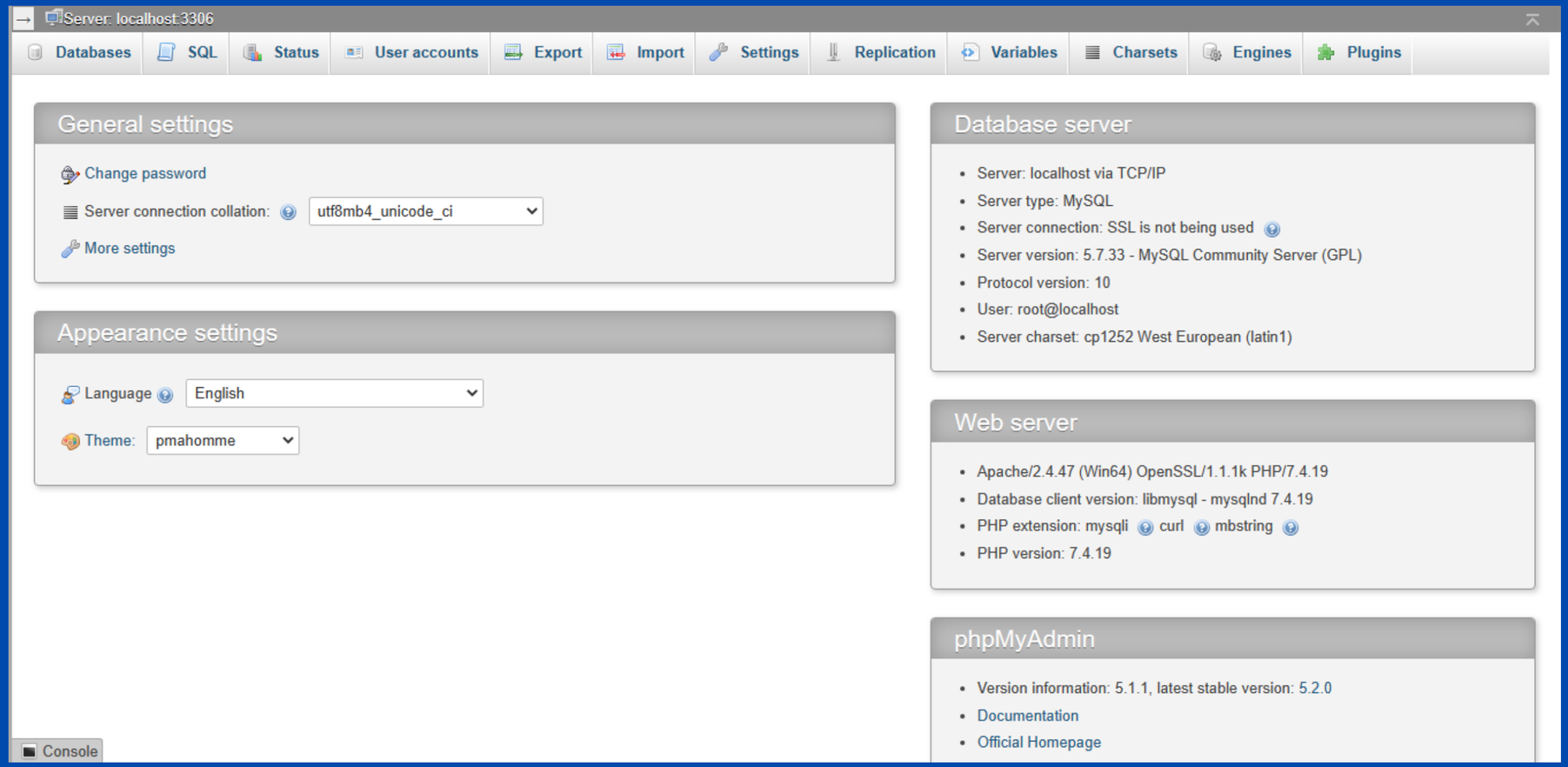
```php
private const DB_PASS = 'password';
```

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

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

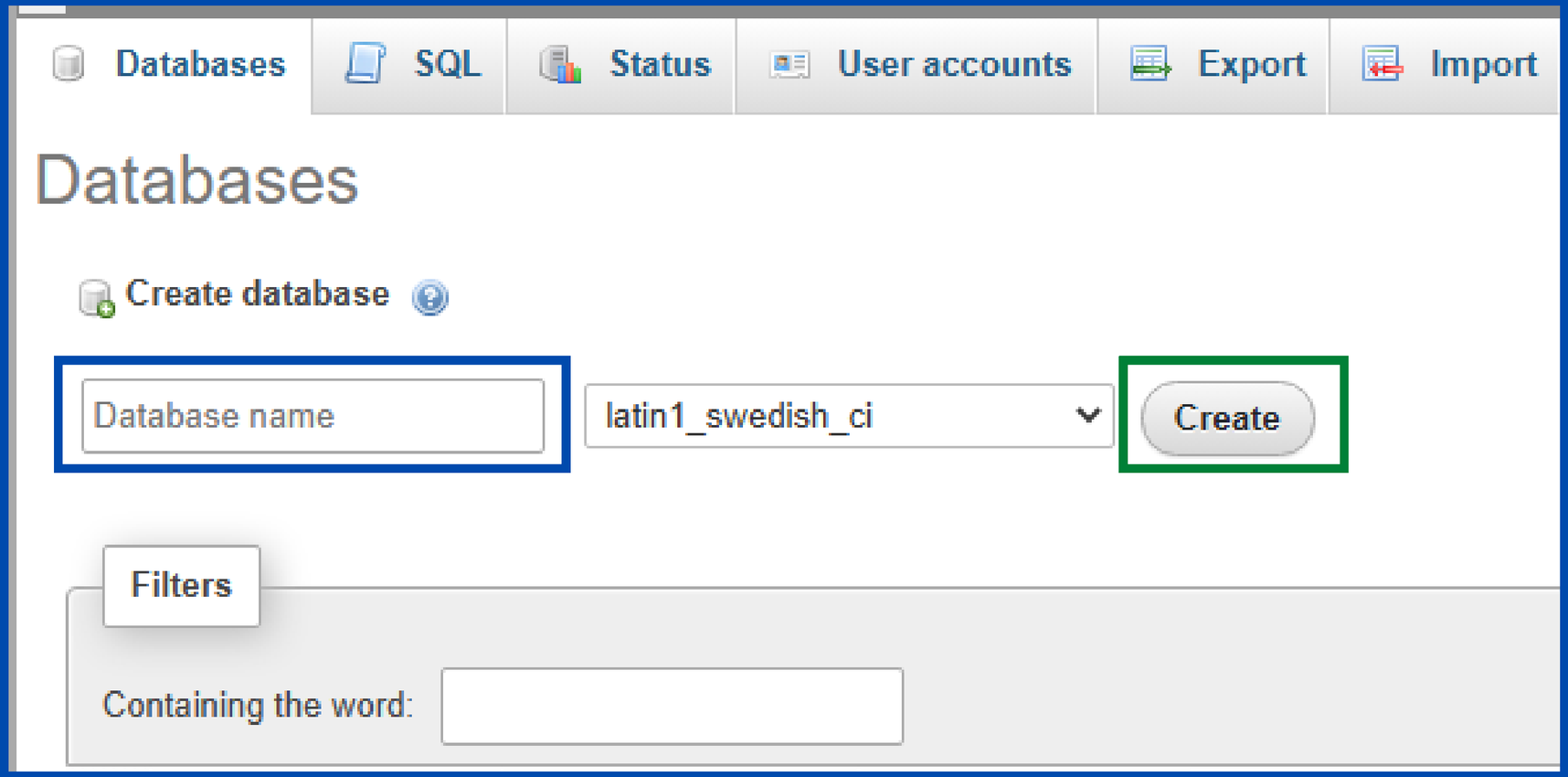
```php
    private const DB_NAME = 'pdo';

```

Akses Database



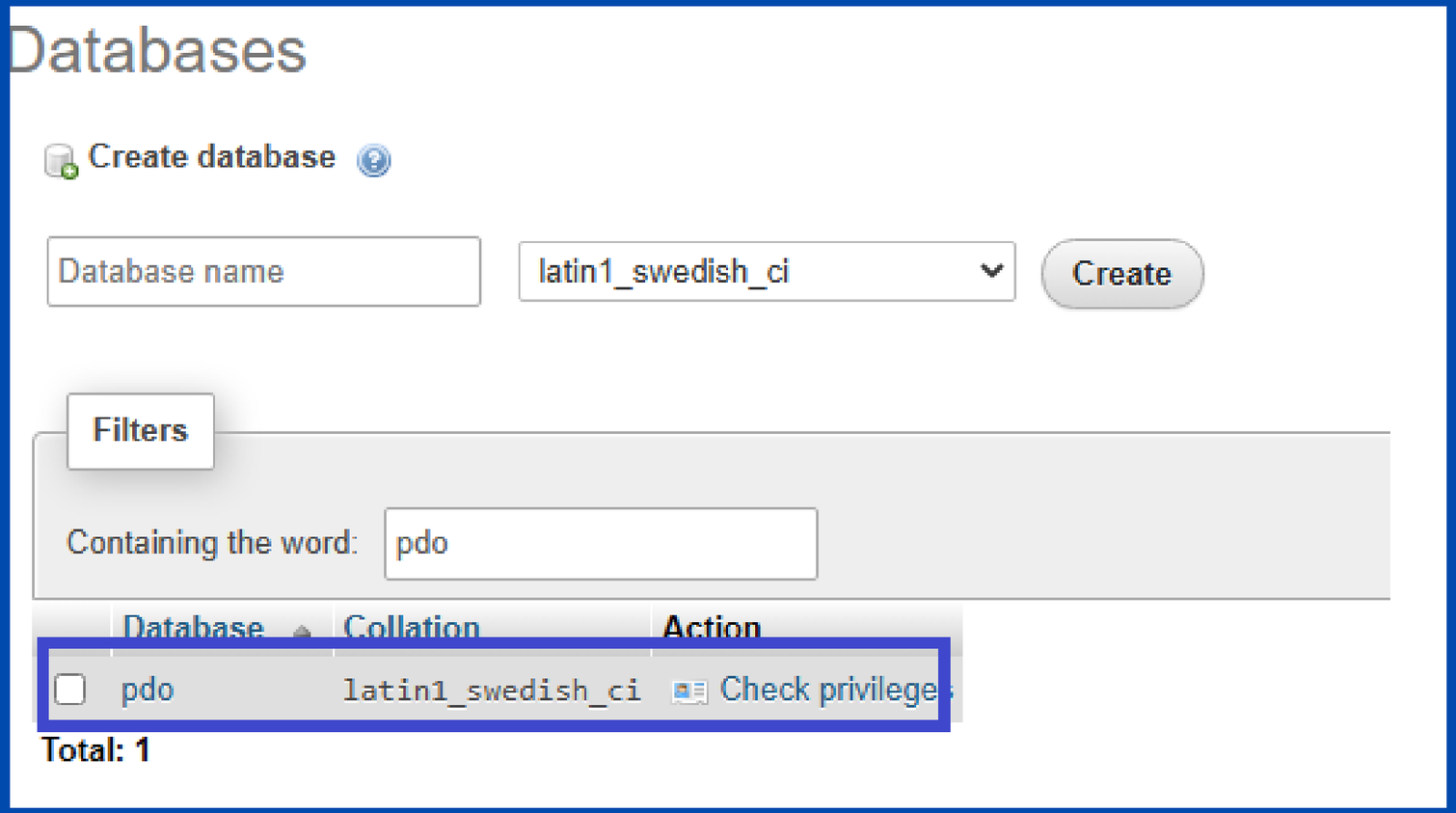
Membuat Database Mysql menggunakan PHPMyAdmin, dengan cara mengakses URL "localhost/phpmyadmin".

Membuat Database



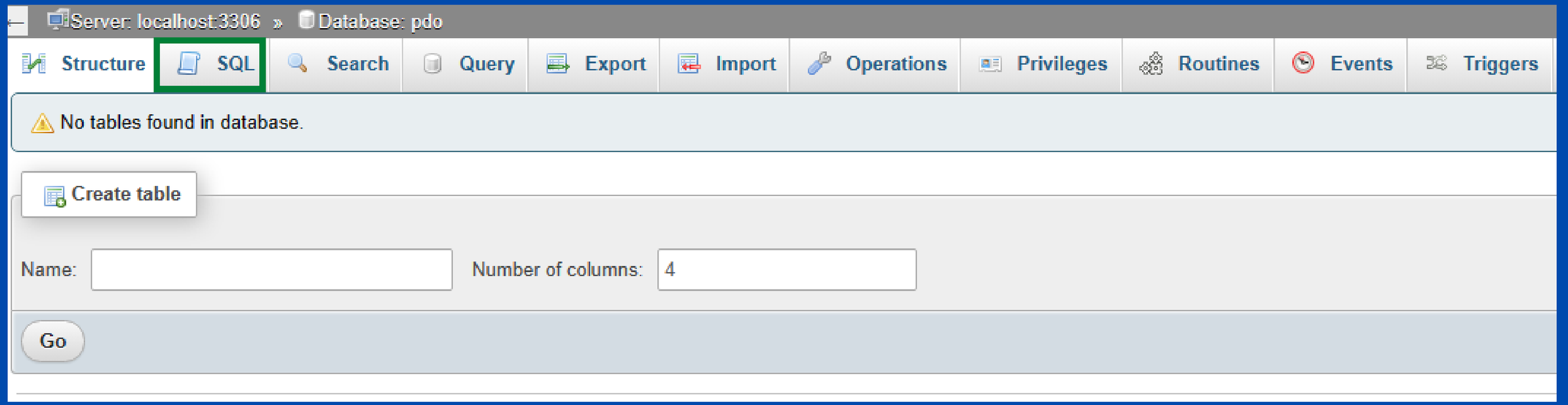
Setelah berada di halaman PHPMyAdmin, kemudian pilih tab Database dan buatlah database sesuai dengan konfigurasi sebelumnya.

Memilih Database



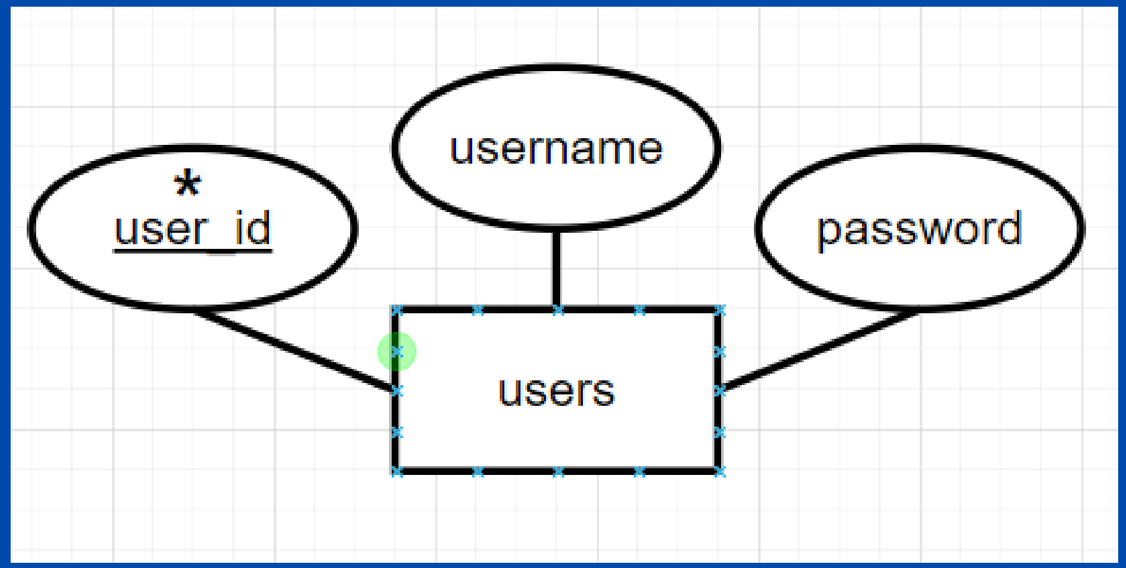
Selanjutnya pilih database yang sudah dibuat dan persiapan selanjutnya yaitu membuat table yang akan dikelola oleh PDO.

Tampilan Halaman Database



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

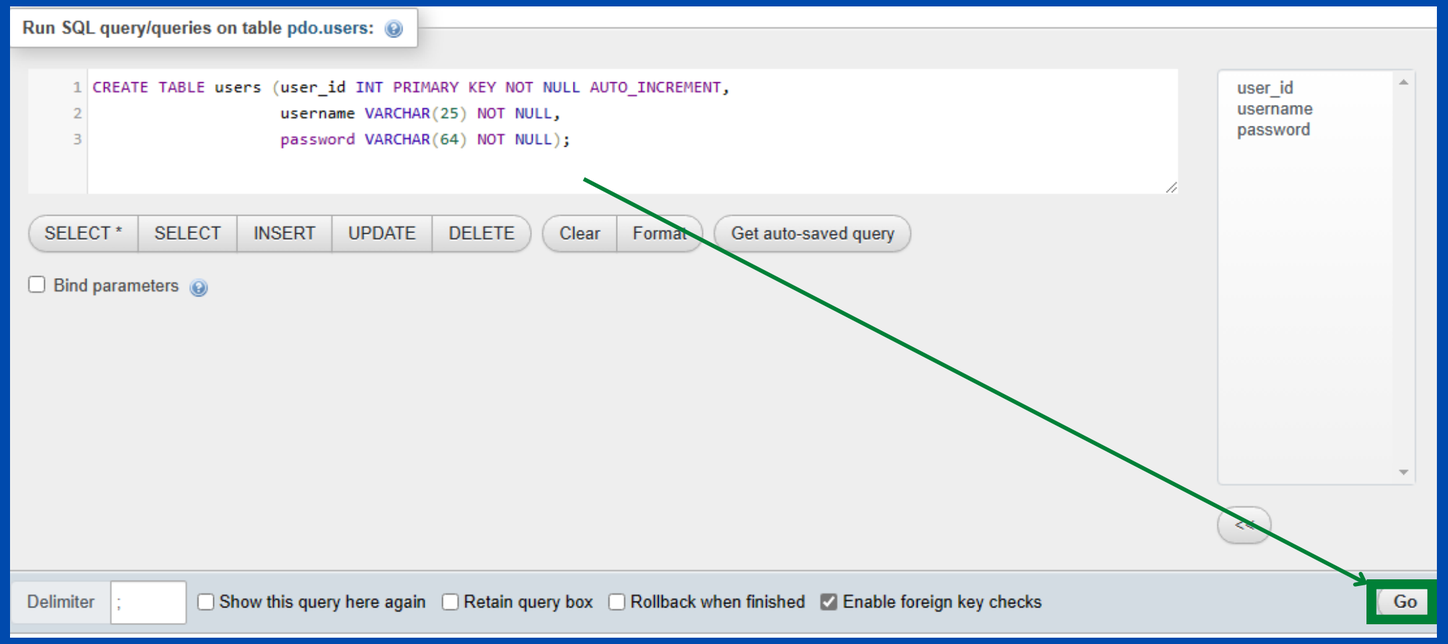
Perancangan Table



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

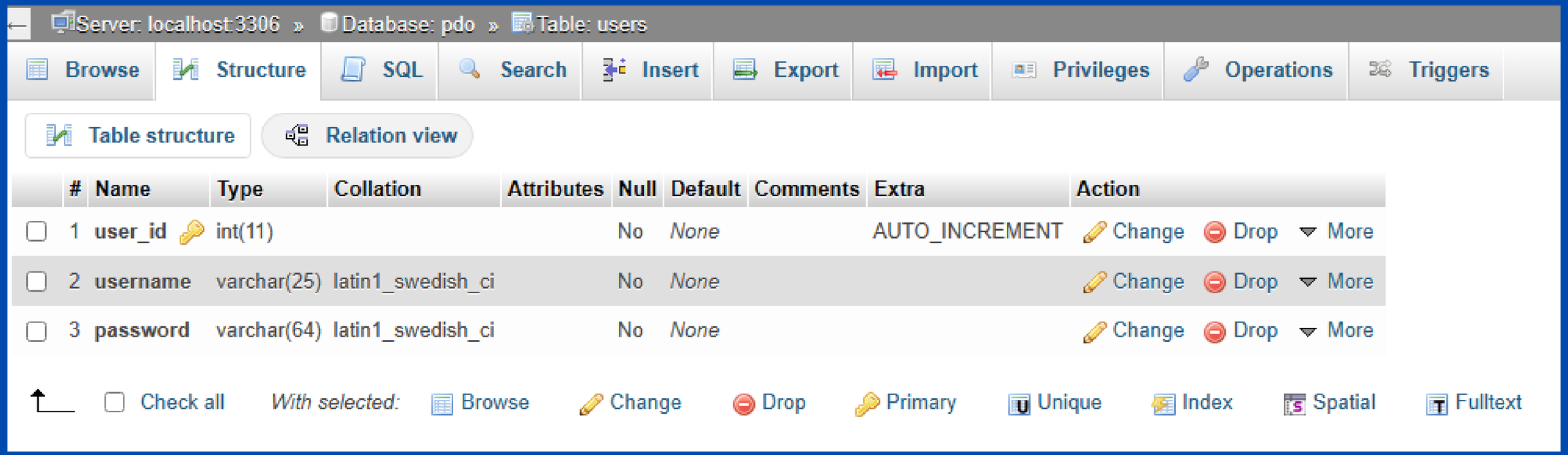
Pembuatan Table




```sql
CREATE TABLE
    users (
        user_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
        username VARCHAR(25) NOT NULL,
        password VARCHAR(64) NOT NULL
    );
```

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

Struktur Data Table



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

Sample Rows Data

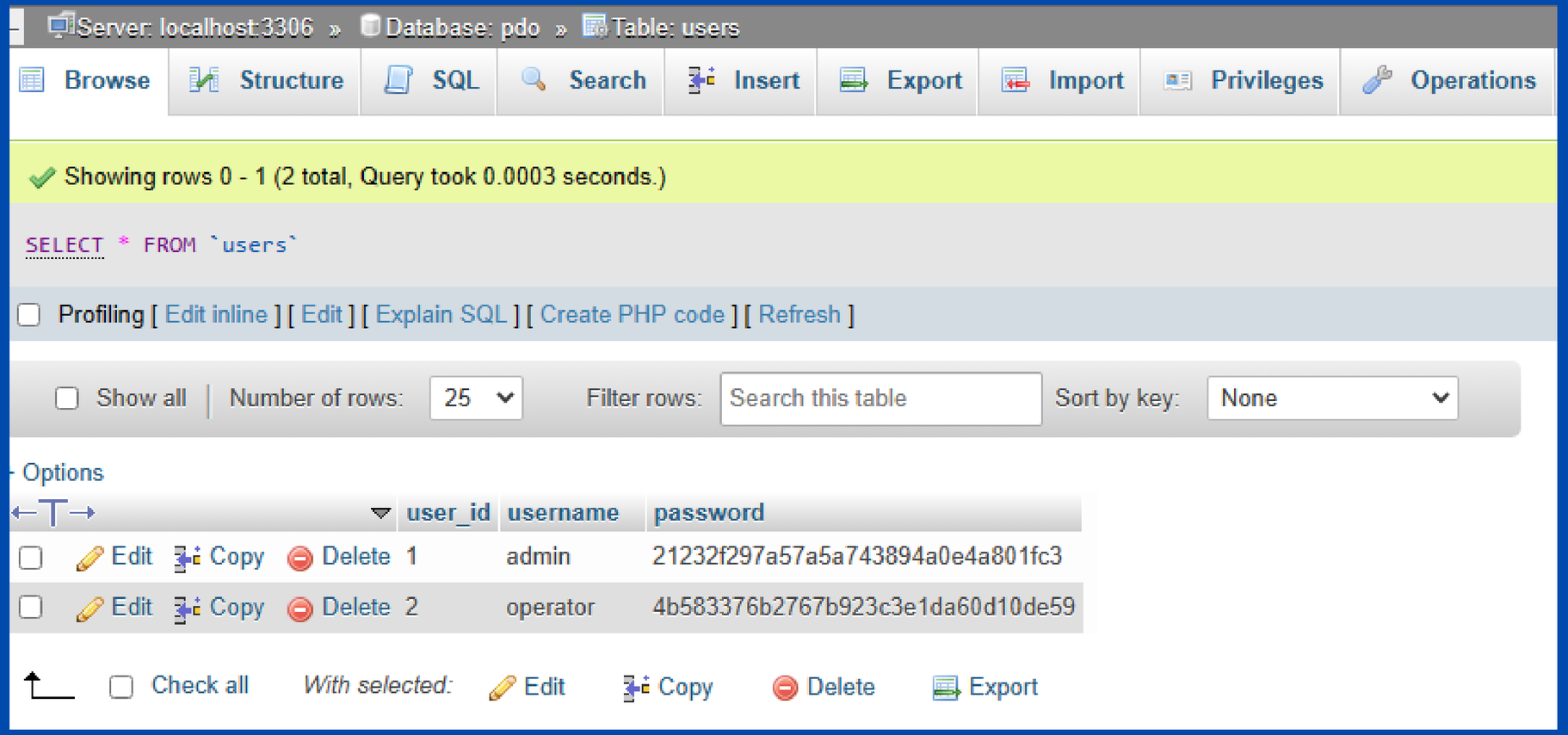
Run SQL query/queries on table pdo.users:

```
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

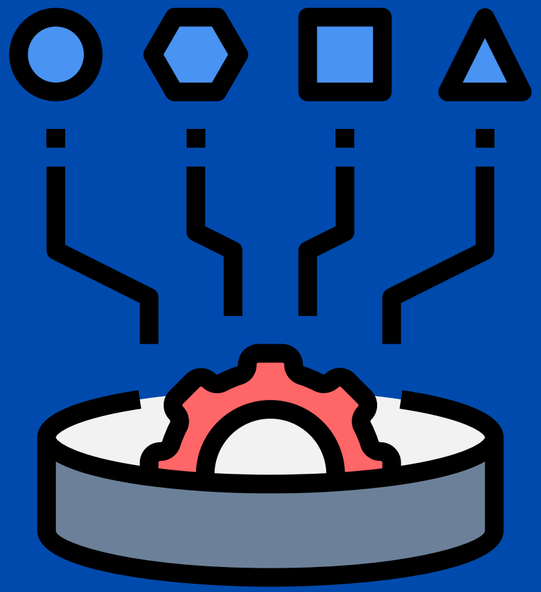
Rows Data



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.



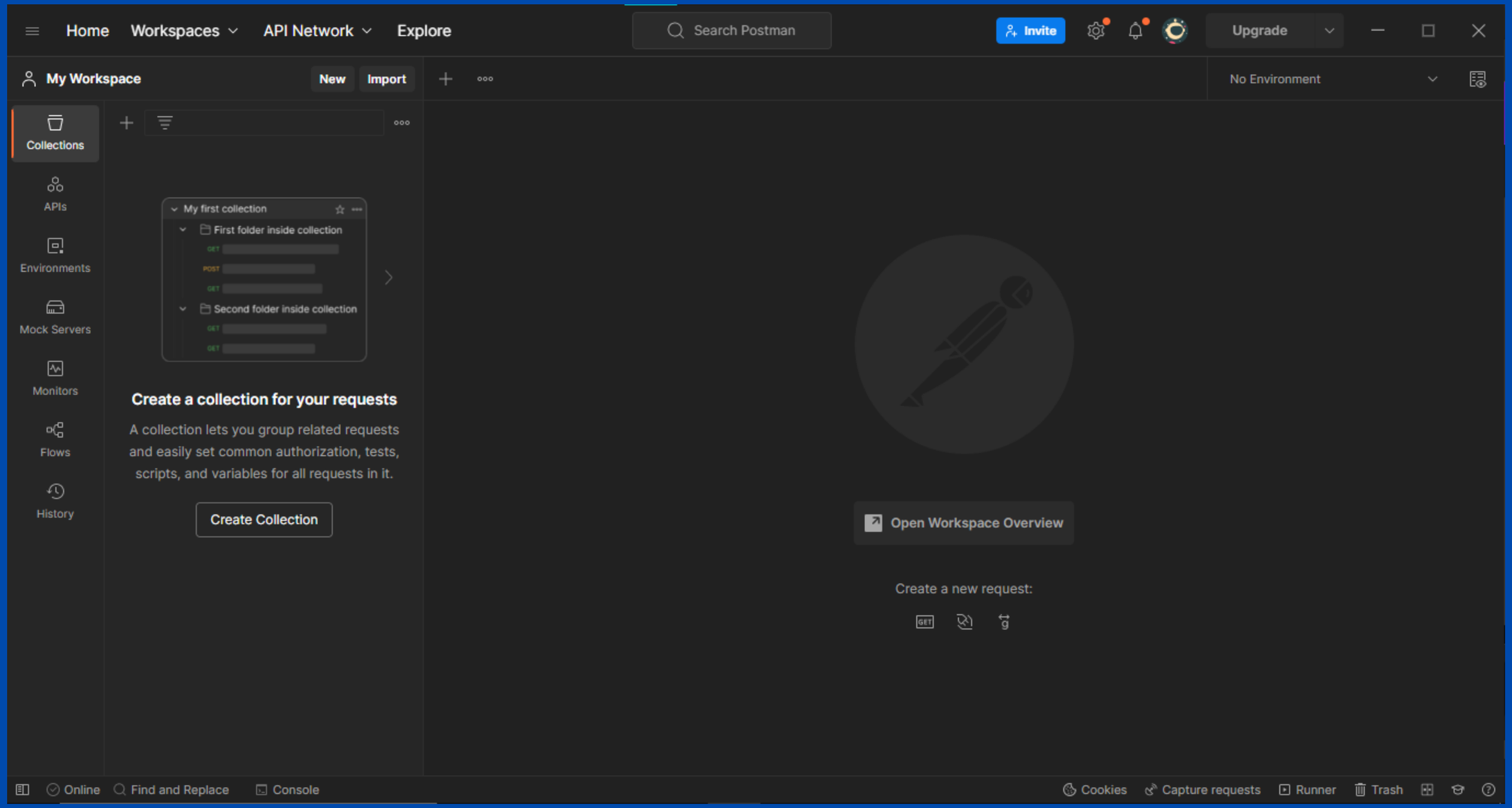
Testing Tools API

Postman

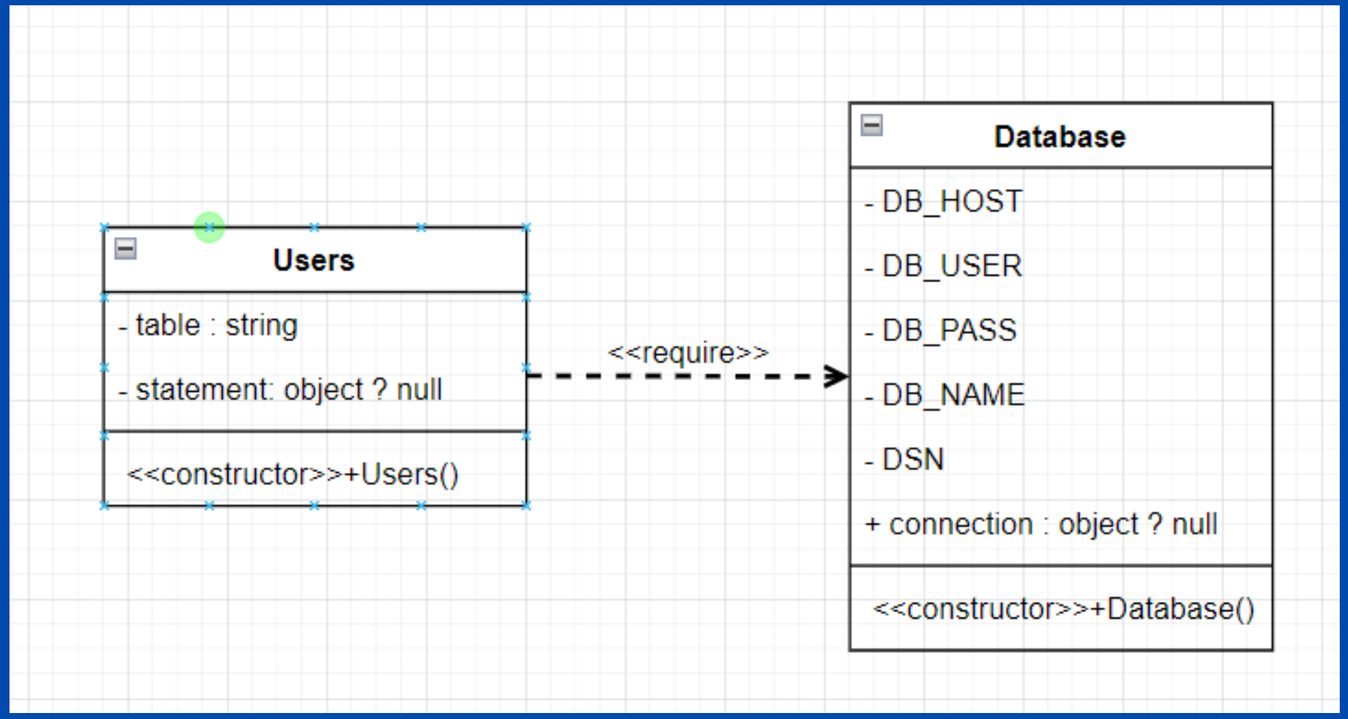
Link Download

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

Tampilan Awal Postman



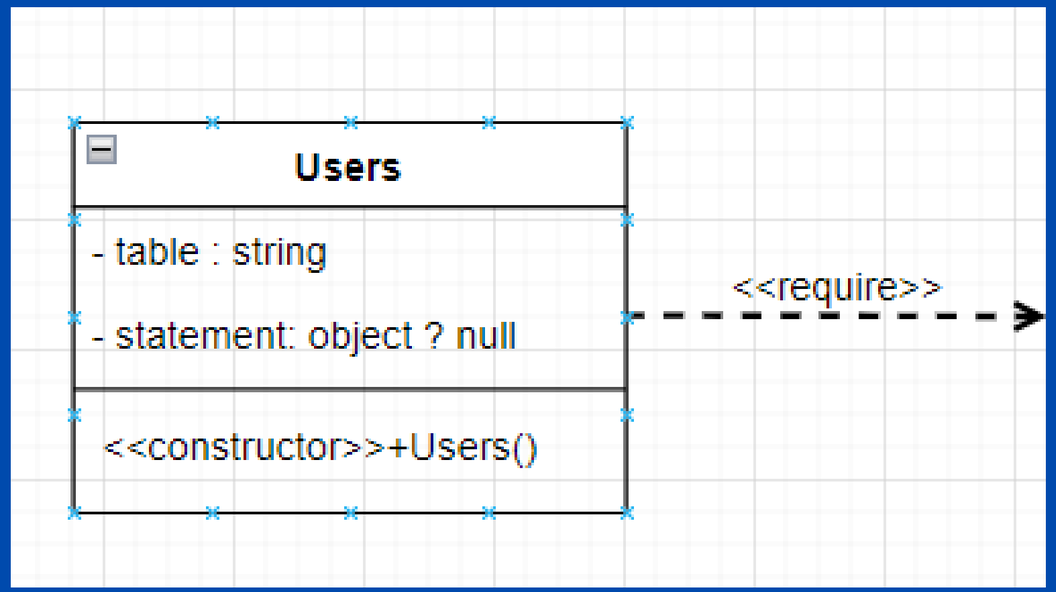
Perancangan Class Diagram Implementasi CRUD



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

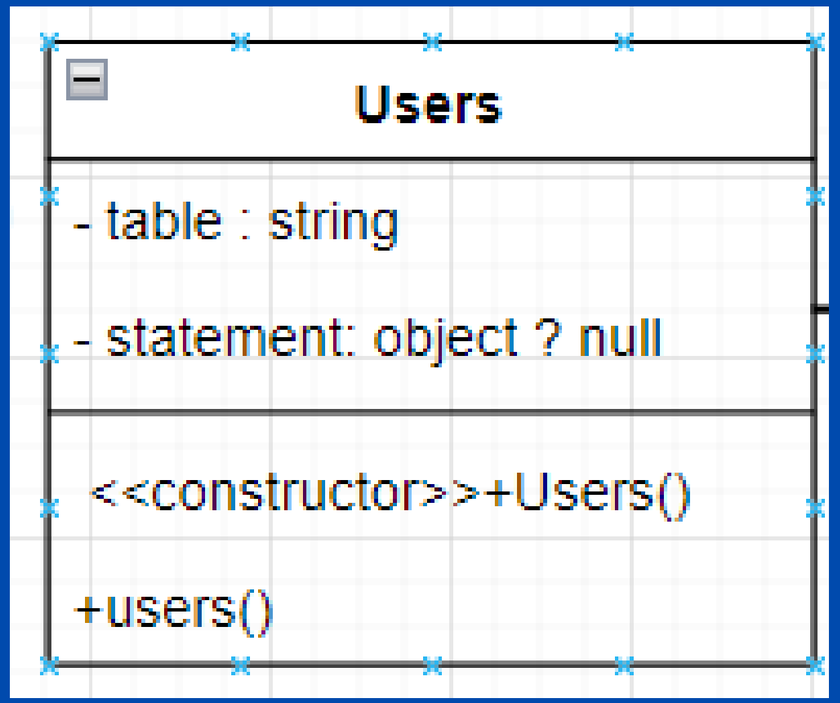
Implementasi CRUD

```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }
}
```



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.

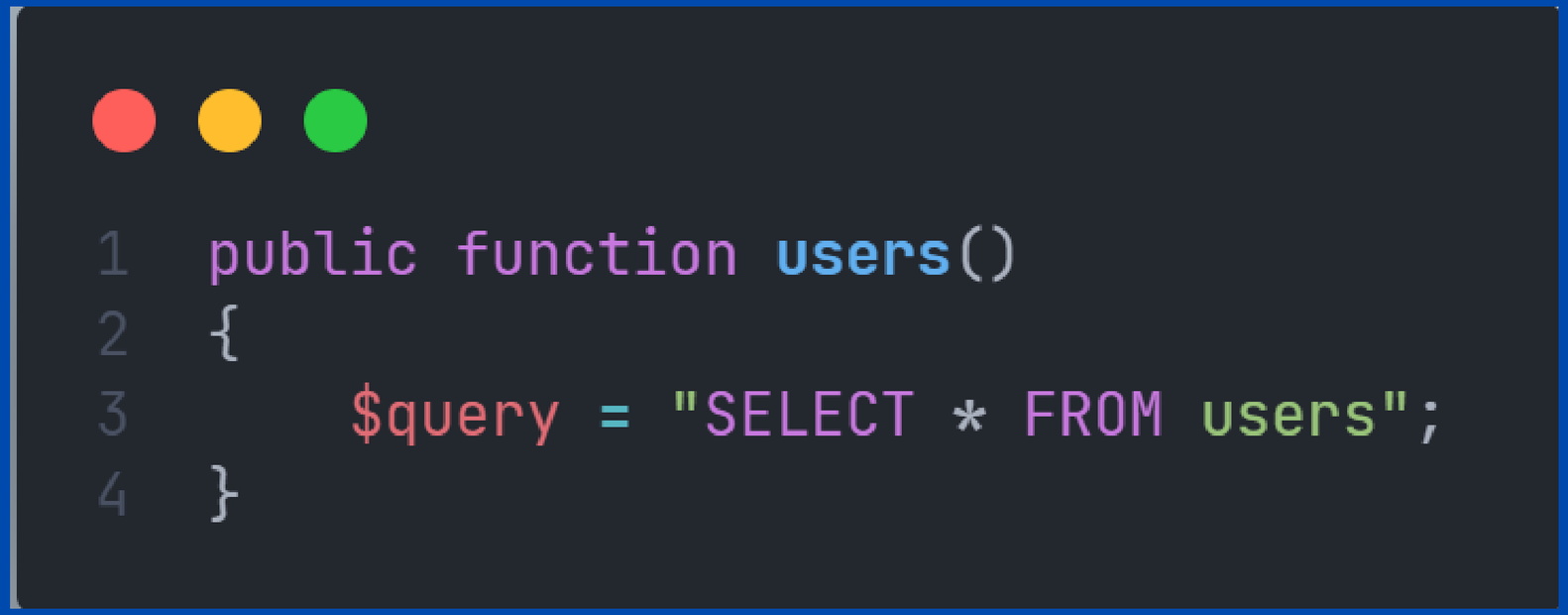
Menampilkan List Data (Read)




```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }

    public function users()
    {
    }
}

```

Untuk menampilkan list data dengan menambahkan method users

Membuat Query PDO



```php
public function users()
{
    $query = "SELECT * FROM users";
}
```

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

Menggunakan Property Select Table

```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }

    public function users()
    {
        $query = "SELECT * FROM $this->table";
    }
}
```

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

Prepare Statement Query

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
}
```

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 tampung terlebih dahulu sebelum dieksekusi query tersebut.

Return Prepare Statement & Debugging

```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }

    public function users()
    {
        $query = "SELECT * FROM $this->table";
        $statement = $this->statement->prepare($query);
        return $statement;
    }
}


$users = new Users();
$users = $users->users();
var_dump($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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    return $statement;
}
```

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

Debugging Execute Statement

```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }

    public function users()
    {
        $query = "SELECT * FROM $this->table";
        $statement = $this->statement->prepare($query);
        $statement->execute();
        var_dump($statement->execute()); die;
        return $statement;
    }
}


$users = new Users();
$users = $users->users();
var_dump($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\ **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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);
    return $statement;
}
```

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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);
    echo "<pre>";
    var_dump($users); die;
    echo "</pre>";
    return $statement;
}
```

```
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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_NUM);
    echo '<pre>';
    print_r($users);
    echo '</pre>';
    return $statement;
}
```

```
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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_NUM);
    foreach ($users as $user) {
        echo 'Username:' . $user[1] . '<br> Password: ' . $user[2] . '<br>';
    }
    return $statement;
}
```

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

PDO::FETCH_ASSOC

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_ASSOC);
    echo "<pre>";
    print_r($users);
    echo "</pre>";
    return $statement;
}
```

```
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.

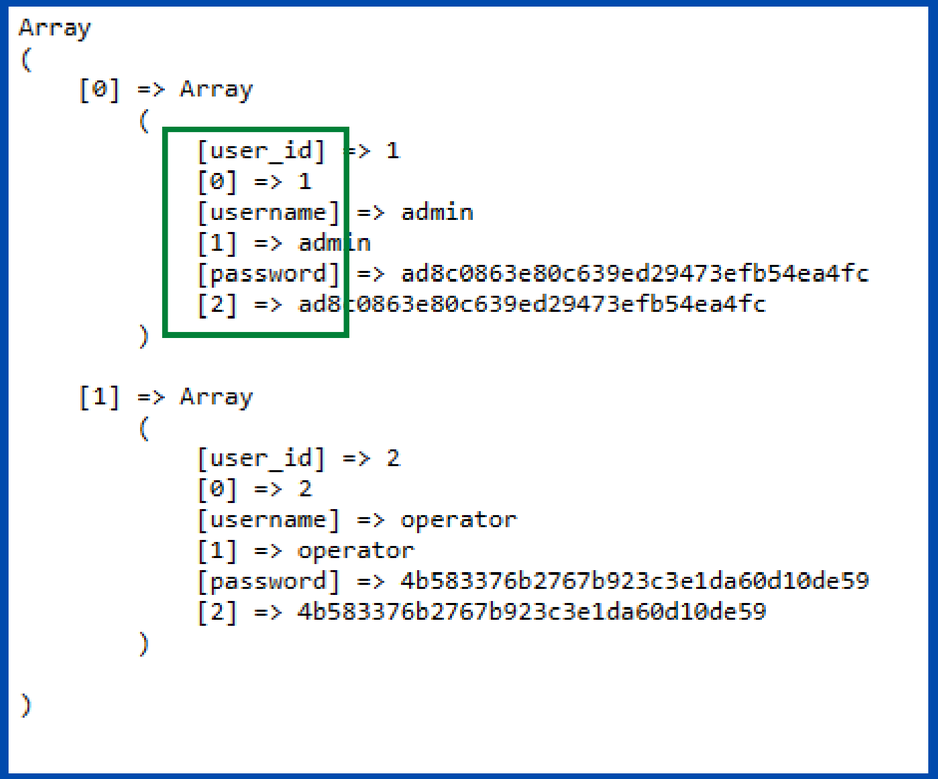
Implementasi PDO::FETCH_ASSOC

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_ASSOC);
    foreach ($users as $user) {
        echo 'Username:' . $user['username'] . '<br> Password: ' . $user['password'] . '<br>';
    }
    return $statement;
}
```

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

PDO::FETCH_BOTH

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_BOTH);
    echo "<pre>";
    print_r($users);
    echo "</pre>";
    return $statement;
}
```



```
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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_BOTH);
    foreach ($users as $user) {
        echo 'Username:' . $user['username'] . '<br> Password: ' . $user['password'] . '<br>';
    }
    return $statement;
}
```

atau

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

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

PDO::FETCH_OBJ

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);
    echo '<pre>';
    print_r($users);
    echo '</pre>';
    return $statement;
}
```

```
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

```php
public function users()
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);

    foreach ($users as $user) {
        echo 'Username:' . $user->username . '<br> Password: ' . $user->password . '<br>';
    }
    return $statement;
}
```

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

API Untuk Menampilkan Data

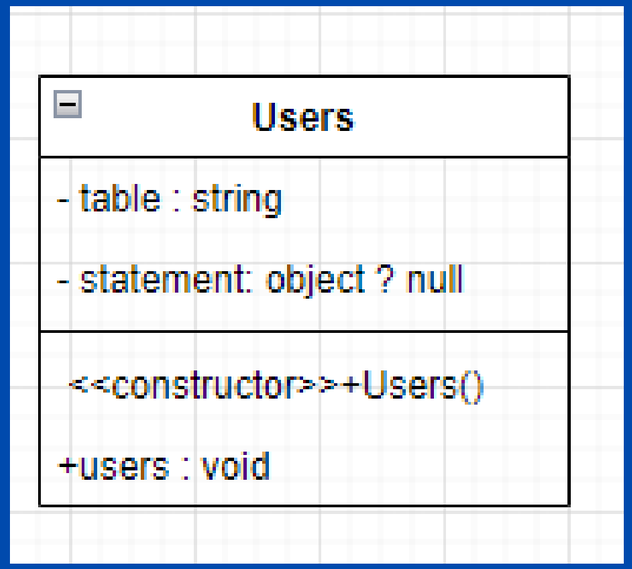
```php
public function users() : void
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);
    $response = [
        'message' => 'Data User Berhasil Ditampilkan',
        'data' => $users
    ];
    echo json_encode($response);
}
```

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

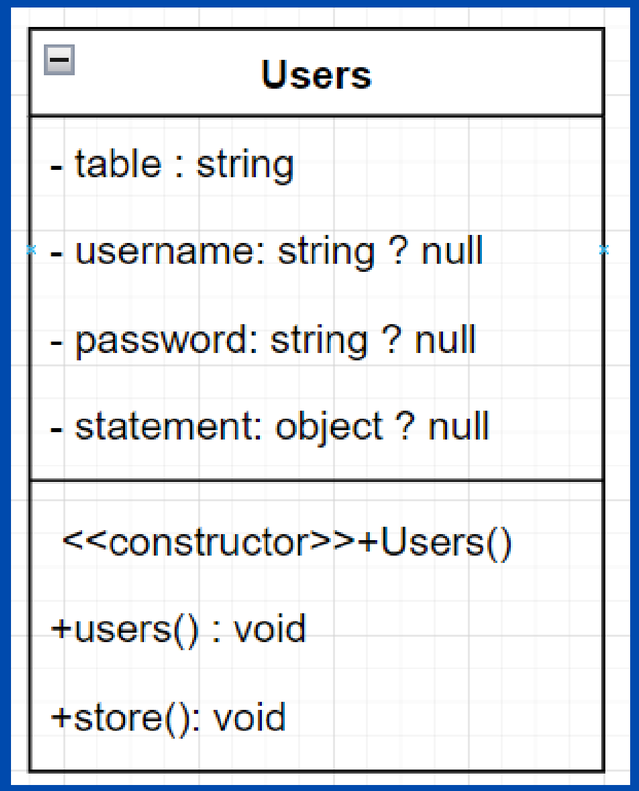
```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
    }

    public function users() : void
    {
        $query = "SELECT * FROM $this->table";
        $statement = $this->statement->prepare($query);
        $statement->execute();
        $users = $statement->fetchAll(PDO::FETCH_OBJ);
        $response = [
            'message' => 'Data User Berhasil Ditampilkan',
            'data' => $users
        ];
        echo json_encode($response);
    }
}


$users = new Users();
$users->users();
```


```json
{
  "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.

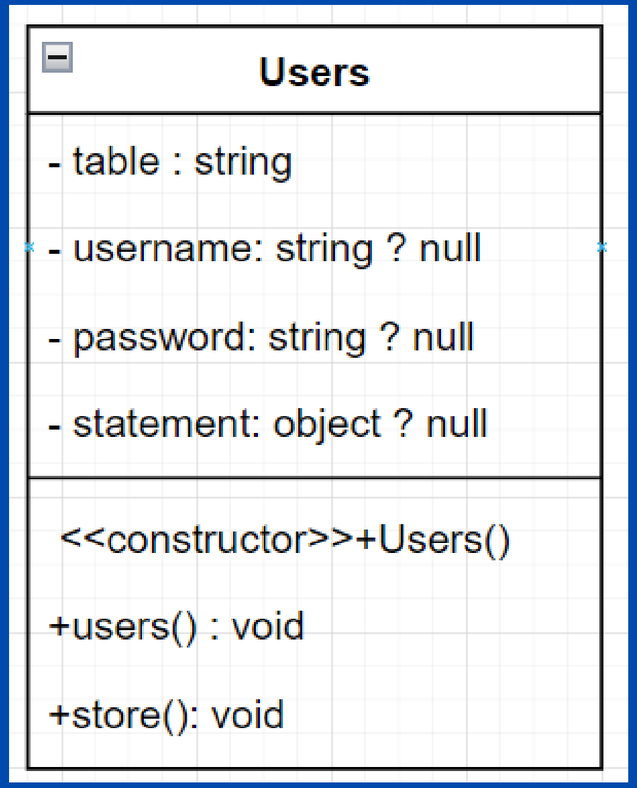
Store / Insert Data (Create)




```php
public function users(): void
{
    $query = "SELECT * FROM $this->table";
    $statement = $this->statement->prepare($query);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_OBJ);
    $response = [
        'message' => 'Data User Berhasil Ditampilkan',
        'data' => $users
    ];
    echo json_encode($response);
}

public function store(): void
{
}
```

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

Properties Untuk Field Data




```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?string $username;
    private ?string $password;
    private ?object $statement;

```

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

Menggunakan Field Pada Constructor

```php
private string $table = 'users';
private ?string $username;
private ?string $password;
private ?object $statement;

public function __construct()
{
    $this->statement = new Database();
    $this->statement = $this->statement->connection;
    $this->username = $_POST['username'] ?? null;
    $this->password = $_POST['password'] ?? null;
}
```

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

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

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

```php
public function store(): void
{
    $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
    $statement = $this->statement->prepare($query);
    $statement->bindParam(':username', $this->username);
    $statement->bindParam(':password', $this->password);
}
```

Statement binding merupakan statement berupa method dari object yang disediakan oleh PDO untuk melakukan bind (mengkaitkan) 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

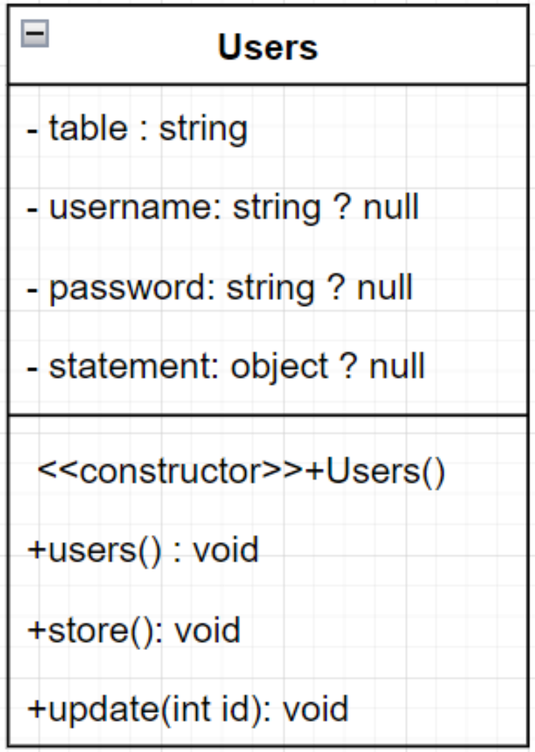
```php
public function insert(): void
{
    $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
    $statement = $this->statement->prepare($query);
    $statement->bindParam(':username', $this->username);
    $statement->bindParam(':password', $this->password);
    $statement->execute();
    $response = [
        'message' => 'Data User Berhasil Ditambahkan',
    ];
    echo json_encode($response);
}
```

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

Implementasi Insert Data

```php
<?php

require_once('Database.php');

class Users
{
    private string $table = 'users';
    private ?object $statement;

    public function __construct()
    {
        $this->statement = new Database();
        $this->statement = $this->statement->connection;
        $this->username = $_POST['username'] ?? null;
        $this->password = $_POST['password'] ?? null;
    }

    public function users(): void
    {
        $query = "SELECT * FROM $this->table";
        $statement = $this->statement->prepare($query);
        $statement->execute();
        $users = $statement->fetchAll(PDO::FETCH_OBJ);
        $response = [
            'message' => 'Data User Berhasil Ditampilkan',
            'data' => $users
        ];
        echo json_encode($response);
    }

    public function store(): void
    {
        $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
        $statement = $this->statement->prepare($query);
        $statement->bindParam(':username', $this->username);
        $statement->bindParam(':password', $this->password);
        $statement->execute();
        $response = [
            'message' => 'Data User Berhasil Dihapus',
        ];
        echo json_encode($response);
    }
}

$users = new Users();
// $users->users();
$users->store();

```

Update Data



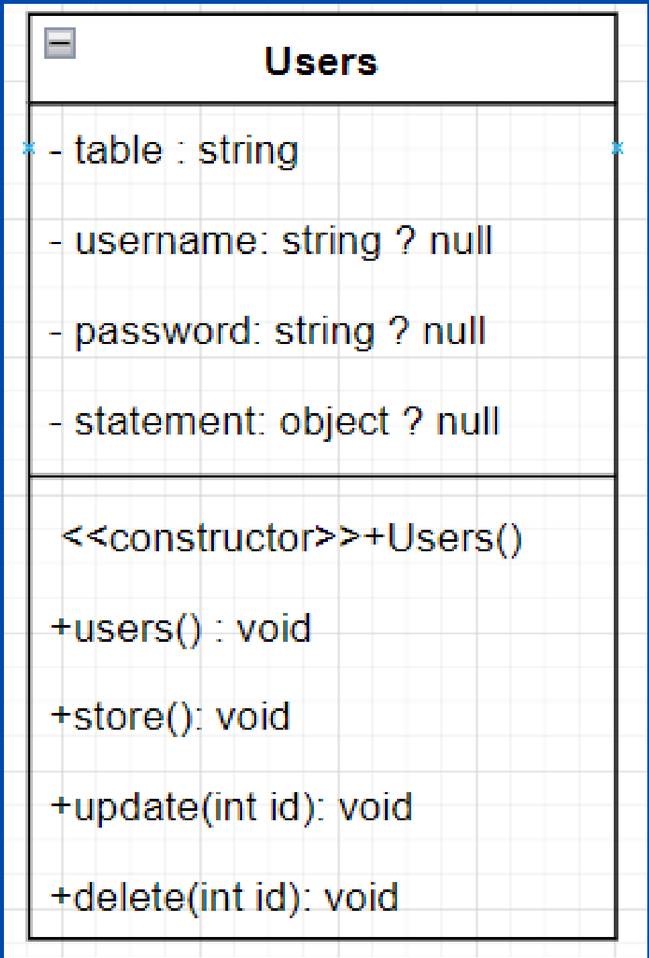

```php
public function update(int $id): void
{
    $query = "UPDATE {$this->table} SET username = :username, password = :password WHERE user_id = :user_id";
    $statement = $this->statement->prepare($query);
    $password = md5($this->password);
    $statement->bindParam(':username', $this->username);
    $statement->bindParam(':password', $password);
    $statement->bindParam(':user_id', $id);
    $statement->execute();
    $response = [
        'message' => 'Data User Berhasil Diubah',
    ];
    echo json_encode($response);
}
```

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




```php
public function delete(int $id): void
{
    $query = "DELETE FROM {$this->table} WHERE user_id = :user_id";
    $statement = $this->statement->prepare($query);
    $statement->bindParam(':user_id', $id);
    $statement->execute();
    $response = [
        'message' => 'Data User Berhasil Dihapus',
    ];
    echo json_encode($response);
}
```

Method yang terakhir adalah menghapus atau delete data.

Implementasi Delete Data

Terimakasih