### 1. Các loại Driver Cơ sở dữ liệu phổ biến trong PHP

**PDO (PHP Data Objects)**

* **Tổng quan:** PDO là một lớp trừu tượng hóa cơ sở dữ liệu, cung cấp một giao diện thống nhất cho việc làm việc với nhiều hệ quản trị cơ sở dữ liệu khác nhau. Điều này có nghĩa là bạn có thể viết mã PHP sử dụng PDO và dễ dàng chuyển đổi giữa các hệ quản trị cơ sở dữ liệu mà không cần phải thay đổi mã nhiều.
* **Driver:** PDO không phải là driver của riêng một hệ quản trị cơ sở dữ liệu nào, mà nó sử dụng các driver cụ thể cho từng hệ quản trị để thực hiện các thao tác. Ví dụ: pdo\_mysql (MySQL), pdo\_pgsql (PostgreSQL), pdo\_sqlite (SQLite).
* **Hỗ trợ:** PDO hỗ trợ MySQL, PostgreSQL, SQLite, Microsoft SQL Server, và nhiều hệ quản trị khác.

**MySQLi (MySQL Improved)**

* **Tổng quan:** MySQLi là một driver chuyên dụng cho MySQL, cung cấp giao diện để kết nối và làm việc với cơ sở dữ liệu MySQL. MySQLi cung cấp cả hai kiểu lập trình thủ tục và hướng đối tượng, và hỗ trợ các tính năng mới của MySQL như chuẩn bị câu truy vấn (prepared statements).
* **Hỗ trợ:** Chỉ hỗ trợ MySQL.

**OCI8**

* **Tổng quan:** OCI8 là một driver chuyên dụng để kết nối PHP với Oracle Database. Driver này hỗ trợ các tính năng đặc biệt của Oracle, như PL/SQL, LOBs, và các giao dịch phức tạp.
* **Hỗ trợ:** Chỉ hỗ trợ Oracle Database.

**SQLSRV**

* · **Tổng quan:** SQLSRV là một driver do Microsoft phát triển để kết nối PHP với Microsoft SQL Server. Nó hỗ trợ các tính năng của SQL Server như chuẩn bị câu truy vấn, giao dịch, và xử lý lỗi.
* **Hỗ trợ:** Chỉ hỗ trợ Microsoft SQL Server.

**SQLite**

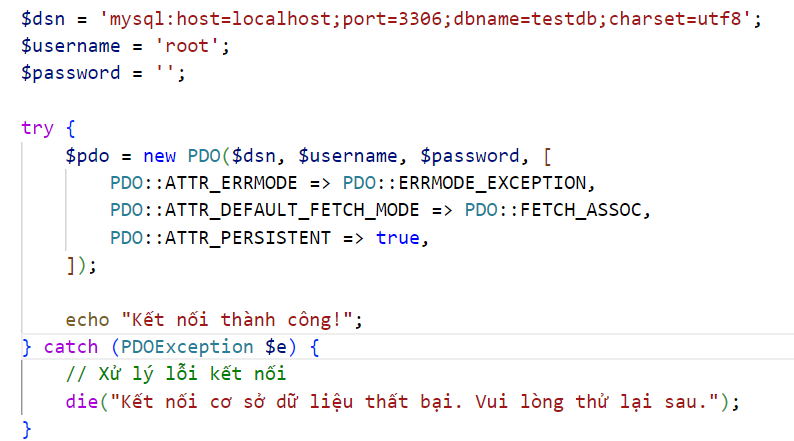
* **Tổng quan:** PHP có hỗ trợ tích hợp cho SQLite, một cơ sở dữ liệu nhúng nhẹ. SQLite không yêu cầu cài đặt một máy chủ cơ sở dữ liệu riêng biệt, vì vậy nó rất tiện lợi cho các ứng dụng nhỏ hoặc phát triển.
* **Hỗ trợ:** Chỉ hỗ trợ SQLite.

### Khi nào sử dụng từng loại driver?

* **PDO:** Sử dụng khi bạn cần viết mã di động, có khả năng làm việc với nhiều hệ quản trị cơ sở dữ liệu khác nhau mà không cần thay đổi nhiều mã nguồn.
* **MySQLi:** Sử dụng khi bạn chỉ làm việc với MySQL và cần tận dụng các tính năng cụ thể của MySQL mà PDO có thể không hỗ trợ đầy đủ.
* **OCI8, SQLSRV, SQLite:** Sử dụng khi bạn cần làm việc với các hệ quản trị cơ sở dữ liệu cụ thể tương ứng như Oracle, SQL Server, hoặc SQLite.

### 2. Kết nối đến cơ sở dữ liệu

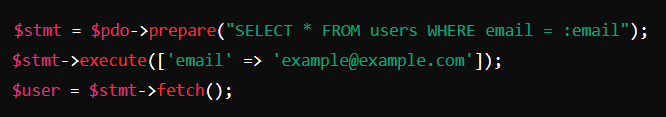
Để kết nối đến cơ sở dữ liệu bằng PDO, bạn cần cung cấp các thông tin cơ bản như DSN (Data Source Name), tên người dùng và mật khẩu.



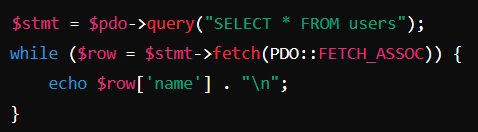
* **$dsn (Data Source Name):** Chuỗi kết nối đến cơ sở dữ liệu chứa thông tin cần thiết để kết nối:
* **mysql:** Chỉ định loại cơ sở dữ liệu là MySQL.
* **host=localhost:** Địa chỉ máy chủ cơ sở dữ liệu. Ở đây là localhost, tức là máy chủ cục bộ.
* **port=3306:** Cổng kết nối MySQL. Ở đây là cổng mặc định 3306.
* **dbname=testdb:** Tên của cơ sở dữ liệu mà bạn muốn kết nối (testdb).
* **charset=utf8:** Thiết lập mã hóa ký tự là UTF-8 để đảm bảo dữ liệu được lưu trữ và xử lý đúng định dạng ký tự.
* **$username:** Tên người dùng để kết nối đến cơ sở dữ liệu. Ở đây là root.
* **$password:** Mật khẩu của người dùng để kết nối đến cơ sở dữ liệu. Ở đây là your\_password.
* **try block:** Khối try được sử dụng để bọc lấy các đoạn mã có thể gây ra ngoại lệ (exception). Nếu một lỗi xảy ra trong khối try, nó sẽ được bắt và xử lý trong khối catch.
* **$pdo = new PDO($dsn, $username, $password, [...]):** Đây là nơi khởi tạo đối tượng PDO để kết nối đến cơ sở dữ liệu. Đối tượng PDO này được tạo với các tham số sau:
* **$dsn:** Chuỗi kết nối chứa thông tin cơ sở dữ liệu (như đã giải thích ở trên).
* **$username và $password:** Thông tin đăng nhập để kết nối đến cơ sở dữ liệu.
* **Các thuộc tính tùy chọn:**
  + **PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION:** Thiết lập chế độ xử lý lỗi. Khi có lỗi xảy ra, nó sẽ ném ra một ngoại lệ (PDOException), giúp dễ dàng xử lý lỗi hơn.
  + **PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC:** Thiết lập chế độ mặc định để lấy dữ liệu từ cơ sở dữ liệu là FETCH\_ASSOC, tức là các dòng kết quả sẽ được trả về dưới dạng mảng liên kết (associative array) với tên cột là khóa của mảng.
  + **PDO::ATTR\_PERSISTENT => true:** Thiết lập kết nối bền vững (persistent connection). Kết nối bền vững giữ kết nối mở lâu hơn thay vì đóng nó sau mỗi yêu cầu, điều này có thể cải thiện hiệu suất.
* **echo "Kết nối thành công!";**: Nếu kết nối thành công, thông báo này sẽ được in ra.
* **catch (PDOException $e):** Nếu có lỗi xảy ra trong khối try, khối catch sẽ bắt ngoại lệ đó. PDOException là lớp ngoại lệ được ném ra khi có lỗi liên quan đến PDO (như kết nối thất bại hoặc lỗi cú pháp SQL).
* **die("Kết nối cơ sở dữ liệu thất bại. Vui lòng thử lại sau.");**: Hàm die() dừng thực thi script và hiển thị thông báo lỗi cho người dùng. Trong trường hợp này, nếu kết nối cơ sở dữ liệu thất bại, thông báo "Kết nối cơ sở dữ liệu thất bại. Vui lòng thử lại sau." sẽ được hiển thị.

1. **Hiểu các thành phần cơ bản của PDO**

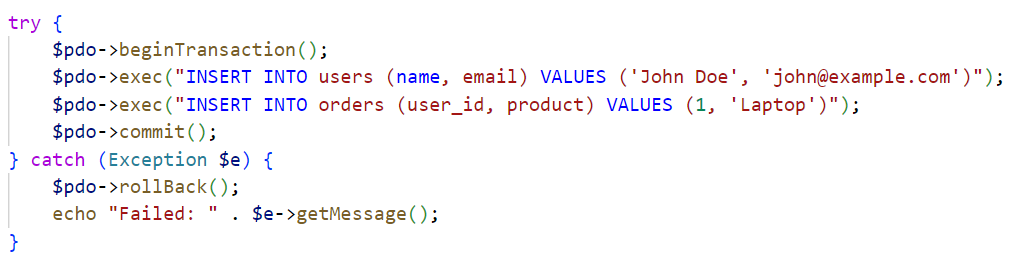
* **$pdo**: Đối tượng PDO đại diện cho kết nối đến cơ sở dữ liệu.
* **Prepared Statements (Câu truy vấn chuẩn bị sẵn)**: Giúp bảo vệ chống lại SQL Injection và cải thiện hiệu suất khi thực hiện nhiều lần truy vấn tương tự.



* **Fetching data**: PDO cung cấp các phương thức như fetch(), fetchAll() để lấy dữ liệu từ kết quả truy vấn.

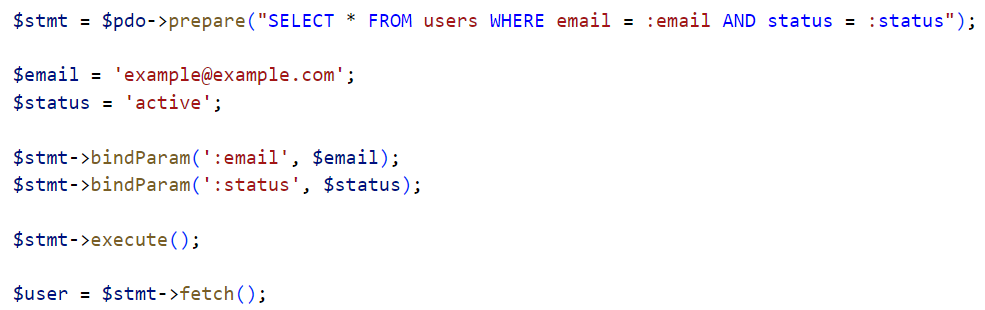


* **Transactions**: PDO hỗ trợ giao dịch để đảm bảo rằng một nhóm các câu lệnh SQL được thực hiện hoàn toàn hoặc không thực hiện gì cả.



### 4. BindParams chống SQL Injection

**SQL Injection** là một trong những lỗ hổng bảo mật phổ biến nhất khi làm việc với cơ sở dữ liệu. PDO giúp bạn chống lại SQL Injection bằng cách sử dụng bindParam() hoặc bindValue() để ràng buộc giá trị vào các tham số trong câu truy vấn chuẩn bị sẵn.



### 5. Tiêu chuẩn đặt tên bảng và trường

**Tên bảng (Table names)**:

* + Tên bảng thường là số nhiều của tên model. Ví dụ:
    - Nếu bạn có một model **User**, tên bảng sẽ là **users**.
    - Nếu bạn có một model **OrderItem**, tên bảng sẽ là **order\_items**
  + Tên bảng thường được viết bằng chữ thường và sử dụng dấu gạch dưới (\_) để ngăn cách các từ. Ví dụ: **order\_items, product\_galleries**.
  + Bảng cho quan hệ nhiều nhiều. Ví dụ bảng **catalogues** quan hệ n-n với **products**. Bảng quan hệ n-n sẽ được thiết kế như sau:
    - Tên bảng: **product\_catalogue**
    - Tên các cột: **product\_id, catalogue\_id**
    - Khóa chính: **Cặp khóa chính** gồm 2 trường **product\_id và catalogue\_id**

**Tên trường (Column names)**:

* + Khóa chính mặc định nên đặt là **id**.
  + Tên trường thường được viết bằng chữ thường, sử dụng dấu gạch dưới (\_) để ngăn cách các từ. Ví dụ: **first\_name, created\_at**.
  + Tên cột thì **KHÔNG** có chứa tên bảng. Ví dụ bảng là customers -> tên trường name chỉ đơn giản là **name**, không đặt là **~~customer\_name~~**
  + Tên cột thì **CÓ** chứa tên bảng chỉ khi giá trị của trường đó nằm ở bảng khác. Ví dụ: bảng orders lưu thông tin địa chỉ từ bảng addresses (city, district,…). Khi tạo trường lưu bên bảng **orders** ta sẽ đặt là **address\_city, address\_district,….**

**Khóa ngoại (Foreign keys)**:

* + Tên khóa ngoại thường là tên của model có liên quan, theo sau là \_id. Ví dụ: nếu một bảng **posts (Post)** có liên kết với bảng **users (User)**, thì khóa ngoại trong posts sẽ là **user\_id**.