PDO: PHP Data Objects

Accesso sicuro ai database MySQL

Prof. Massimo Fedeli

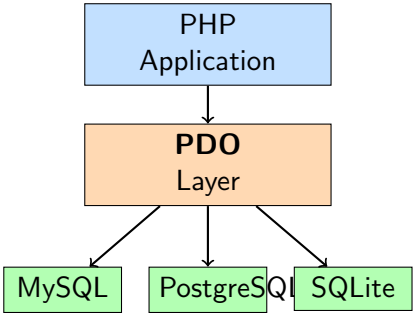
IIS Fermi Sacconi Ceci - Ascoli Piceno

December 16, 2025

Cos'è PDO?

PHP Data Objects (PDO) è un'estensione che fornisce:

- Interfaccia **unificata** per l'accesso ai database
- Supporto per **12+ database** diversi (MySQL, PostgreSQL, SQLite, Oracle, etc.)
- Meccanismi di **sicurezza** integrati
- **Prepared statements** nativi
- Gestione avanzata degli **errori**



Perché usare PDO?

✓ Vantaggi di PDO

1. **Sicurezza**: protezione nativa contro SQL Injection
2. **Portabilità**: codice riutilizzabile su diversi DBMS
3. **Prestazioni**: prepared statements più veloci per query ripetute
4. **Flessibilità**: modalità di fetch personalizzabili
5. **Transazioni**: supporto completo per BEGIN, COMMIT, ROLLBACK
6. **OOP**: approccio orientato agli oggetti

PDO vs mysqli

Caratteristica	**PDO**	**mysqli**
Database supportati	12+	Solo MySQL
API	Solo OOP	OOP + Procedurale
Named parameters	✓	×
Prepared statements	✓	✓
Portabilità	Eccellente	Limitata
Curva apprendimento	Media	Bassa
Performance	Ottima	Ottima

Raccomandazione: PDO per nuovi progetti per maggiore flessibilità

Connessione Base

```php
<?php
// Parametri di connessione
$host = 'localhost';
$dbname = 'scuola';
$username = 'root';
$password = '';

try {
  // Creazione oggetto PDO
  $pdo = new PDO(
  "mysql:host=$host;dbname=$dbname;charset=utf8mb4",
  $username,
  $password
  );

  echo "Connessione riuscita!";

} catch (PDOException $e) {
  die("Errore connessione: " . $e->getMessage());
```

Anatomia del DSN

DSN - Data Source Name

```
mysql:host=localhost;dbname=scuola;charset=utf8mb4
```

`mysql:`	Driver del database (mysql, pgsql, sqlite, etc.)
`host=`	Indirizzo del server database
`dbname=`	Nome del database
`charset=`	Codifica caratteri (sempre utf8mb4 per MySQL)

Altri parametri opzionali:

- `port=3306` - Porta personalizzata
- `unix_socket=/path/to/socket` - Socket Unix

Opzioni di Connessione

```php
<?php
$options = [
// Modalita' errore: eccezioni
PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,

// Fetch mode predefinito
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,

// Disabilita emulazione prepared statements
PDO::ATTR_EMULATE_PREPARES => false,

// Connessione persistente (opzionale)
PDO::ATTR_PERSISTENT => false
];

$pdo = new PDO($dsn, $username, $password, $options);
?>
```

Gestione Errori: Le Tre Modalità

1. ERRMODE_SILENT

```php
// Nessun errore mostrato
// Controllo manuale
if (!$stmt->execute()) {
  print_r($stmt->errorInfo());
}
```

2. ERRMODE_WARNING

```php
// Warning PHP standard
// Non interrompe esecuzione
```

3. ERRMODE_EXCEPTION ✓

```php
try {
  $stmt = $pdo->query($sql);
} catch (PDOException $e) {
  echo "Errore: ";
  echo $e->getMessage();
}
```

Best Practice: Usare sempre `EXCEPTION`

Chiusura Connessione

```php
<?php
// Connessione aperta
$pdo = new PDO($dsn, $username, $password);

// ... operazioni sul database ...

// Chiusura esplicita
$pdo = null;

// La connessione si chiude automaticamente
// alla fine dello script
?>
```

[i] **Nota**: PHP chiude automaticamente le connessioni, ma è buona pratica chiuderle esplicitamente quando non servono più.

Query Semplice con query()

Metodo `query()`: per query senza parametri

```php
<?php
$sql = "SELECT * FROM studenti";

// Esecuzione diretta
$stmt = $pdo->query($sql);

// $stmt e' un oggetto PDOStatement
echo "Righe trovate: " . $stmt->rowCount();
?>

```

[!] **Attenzione**: Mai usare `query()` con dati utente! Rischio SQL Injection!

exec() per INSERT, UPDATE, DELETE

```php
<?php
// exec() restituisce il numero di righe affette
$sql = "DELETE FROM studenti WHERE voto < 6";
$righe = $pdo->exec($sql);

echo "Eliminate $righe righe";

// Altro esempio
$sql = "UPDATE studenti SET classe = '5A'
WHERE classe = '4A'";
$righe = $pdo->exec($sql);

echo "Aggiornate $righe righe";
?>
```

Differenza:

- `query()`: restituisce PDOStatement (per SELECT)
- `exec()`: restituisce numero righe (per INSERT/UPDATE/DELETE)

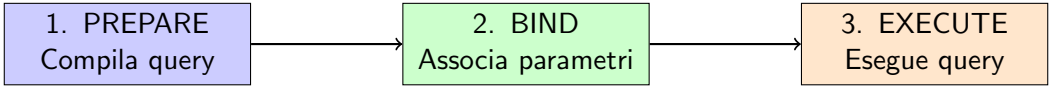
Recupero Ultimo ID Inserito

```php
<?php
$sql = "INSERT INTO studenti (nome, cognome, classe)
VALUES ('Mario', 'Rossi', '3A')";

$pdo->exec($sql);

// Ottieni l'ID auto-incrementato
$ultimoId = $pdo->lastInsertId();

echo "Nuovo studente inserito con ID: $ultimoId";
?>
```

Utile per:

- Relazioni tra tabelle
- Conferme all'utente
- Logging

Cos'è un Prepared Statement?



Query preparata una volta, eseguita più volte con parametri diversi

Vantaggi:

- **Sicurezza**: Previene SQL Injection
- **Performance**: Query pre-compilata
- **Chiarezza**: Codice più leggibile

prepare() ed execute()

Sintassi base:

```php
<?php
// 1. PREPARE - con placeholder ?
$sql = "SELECT * FROM studenti WHERE classe = ?";
$stmt = $pdo->prepare($sql);

// 2. EXECUTE - passa parametri
$stmt->execute(['3A']);

// 3. FETCH - recupera risultati
$risultati = $stmt->fetchAll();

foreach ($risultati as $studente) {
  echo $studente['nome'] . " " . $studente['cognome'];
}
?>
```

Placeholder Posizionali (?)

```php
<?php
$sql = "SELECT * FROM studenti
WHERE classe = ? AND voto >= ?";

$stmt = $pdo->prepare($sql);

// L'ordine dei parametri DEVE corrispondere
$stmt->execute(['3A', 7]);

// Esempio con INSERT
$sql = "INSERT INTO studenti (nome, cognome, classe)
VALUES (?, ?, ?)";
$stmt = $pdo->prepare($sql);
$stmt->execute(['Mario', 'Rossi', '3A']);
?>
```

Pro: Più compatti

Contro: Ordine critico, meno leggibile

Placeholder Nominativi (:nome)

```php
<?php
$sql = "SELECT * FROM studenti
WHERE classe = :classe AND voto >= :voto_min";

$stmt = $pdo->prepare($sql);

// Array associativo - ordine non importante!
$stmt->execute([
':classe' => '3A',
':voto_min' => 7
]);

// Oppure senza i due punti nelle chiavi
$stmt->execute([
'classe' => '3A',
'voto_min' => 7
]);
?>
```

execute() con Array

```php
<?php
// INSERT multipli
$sql = "INSERT INTO studenti (nome, cognome, classe)
VALUES (:nome, :cognome, :classe)";
$stmt = $pdo->prepare($sql);

$studenti = [
['nome' => 'Mario', 'cognome' => 'Rossi', 'classe' => '3A'],
['nome' => 'Laura', 'cognome' => 'Bianchi', 'classe' => '3A'],
['nome' => 'Giuseppe', 'cognome' => 'Verdi', 'classe' => '3B']
];

foreach ($studenti as $studente) {
  $stmt->execute($studente);
}

echo "Inseriti " . count($studenti) . " studenti";
?>
```

bindValue() - Base

Associa un valore a un parametro:

```php
<?php
$sql = "SELECT * FROM studenti WHERE classe = :classe";
$stmt = $pdo->prepare($sql);

// Binding singolo
$stmt->bindValue(':classe', '3A');
$stmt->execute();

// Equivalente a:
// $stmt->execute([':classe' => '3A']);
?>
```

Utile quando:

- Si ha bisogno di controllo fine sui tipi
- Si costruisce la query dinamicamente

bindParam() vs bindValue()

bindValue() - per valore

```php
$classe = '3A';
$stmt->bindValue(':c', $classe);
$classe = '3B'; // Non influisce
$stmt->execute();
// Query con '3A'

```

bindParam() - per riferimento

```php
$classe = '3A';
$stmt->bindParam(':c', $classe);
$classe = '3B'; // Cambia!
$stmt->execute();
// Query con '3B'

```

bindParam() passa la variabile per riferimento - il valore viene letto al momento di `execute()`!

bindParam() - Esempio Pratico

```php
<?php
$sql = "INSERT INTO log (utente, azione, timestamp)
VALUES (:utente, :azione, :ts)";
$stmt = $pdo->prepare($sql);

// Binding per riferimento
$stmt->bindParam(':utente', $utente);
$stmt->bindParam(':azione', $azione);
$stmt->bindParam(':ts', $timestamp);

// Esecuzione multipla con valori diversi
$utente = 'mario'; $azione = 'login';
$timestamp = time();
$stmt->execute();

$utente = 'laura'; $azione = 'logout';
$timestamp = time();
$stmt->execute();
?>
```

Tipi di Dati (PDO::PARAM_*)

```php
<?php
$sql = "SELECT * FROM studenti
WHERE voto >= :voto AND attivo = :attivo";
$stmt = $pdo->prepare($sql);

// Specificare il tipo esplicitamente
$stmt->bindValue(':voto', 6, PDO::PARAM_INT);
$stmt->bindValue(':attivo', true, PDO::PARAM_BOOL);

$stmt->execute();
?>
```

Tipi disponibili:

- `PDO::PARAM_INT` - Intero
- `PDO::PARAM_BOOL` - Booleano
- `PDO::PARAM_STR` - Stringa (default)
- `PDO::PARAM_NULL` - NULL

fetch() - Recupero Riga Singola

```php
<?php
$sql = "SELECT * FROM studenti WHERE id = :id";
$stmt = $pdo->prepare($sql);
$stmt->execute(['id' => 1]);

// Recupera UNA riga
$studente = $stmt->fetch();

if ($studente) {
  echo $studente['nome'] . " " . $studente['cognome'];
} else {
  echo "Studente non trovato";
}
?>
```

Comportamento:

- Prima chiamata: prima riga
- Seconda chiamata: seconda riga

fetch() in Loop

```php
<?php
$sql = "SELECT nome, cognome, voto FROM studenti
WHERE classe = '3A'";
$stmt = $pdo->query($sql);

// Ciclo su tutte le righe
while ($riga = $stmt->fetch()) {
  echo $riga['nome'] . " " . $riga['cognome'];
  echo " - Voto: " . $riga['voto'] . "<br>";
}
?>
```

Vantaggio: Consuma poca memoria anche con molti risultati
Quando usarlo: Elaborazione riga per riga, grandi dataset

fetchAll() - Tutte le Righe

```php
<?php
$sql = "SELECT * FROM studenti WHERE classe = '3A'";
$stmt = $pdo->query($sql);

// Recupera TUTTE le righe in un array
$studenti = $stmt->fetchAll();

echo "Trovati " . count($studenti) . " studenti<br>";

foreach ($studenti as $studente) {
  echo $studente['nome'] . "<br>";
}
?>
```

Quando usarlo:

- Risultati piccoli/medi
- Serve contare le righe prima di elaborarle
- Serve accesso random ai risultati

Modalità di Fetch

PDO::FETCH_ASSOC - Array associativo

```
1       $row = $stmt->fetch(PDO::FETCH_ASSOC);
2       // ['nome' => 'Mario', 'cognome' => 'Rossi']
3
```

PDO::FETCH_NUM - Array numerico

```
1       $row = $stmt->fetch(PDO::FETCH_NUM);
2       // [0 => 'Mario', 1 => 'Rossi']
3
```

PDO::FETCH_BOTH - Entrambi (default)

```
1       $row = $stmt->fetch(PDO::FETCH_BOTH);
2       // ['nome' => 'Mario', 0 => 'Mario', 'cognome' => 'Rossi', 1 => 'Rossi']
3
```

Raccomandazione: Usare `FETCH_ASSOC` per chiarezza

FETCH_OBJ - Oggetto

```php
<?php
$sql = "SELECT * FROM studenti WHERE id = 1";
$stmt = $pdo->query($sql);

// Recupera come oggetto stdClass
$studente = $stmt->fetch(PDO::FETCH_OBJ);

echo $studente->nome;        // Propriet , non chiavi!
echo $studente->cognome;
echo $studente->classe;
?>
```

Utile per:

- Accesso a proprietà con -> invece di []
- Integrazione con codice OOP
- IDE con autocompletamento

FETCH_CLASS - Oggetto Personalizzato

```php
<?php
class Studente {
  public $id;
  public $nome;
  public $cognome;

  public function nomeCompleto() {
    return $this->nome . " " . $this->cognome;
  }
}

$sql = "SELECT * FROM studenti WHERE classe = '3A'";
$stmt = $pdo->query($sql);

// Popola oggetti della classe Studente
$studenti = $stmt->fetchAll(PDO::FETCH_CLASS, 'Studente');

foreach ($studenti as $s) {
  echo $s->nomeCompleto();  // Usa metodo della classe!
```

fetchColumn() - Singola Colonna

```php
<?php
// Recupera solo una colonna
$sql = "SELECT COUNT(*) FROM studenti";
$stmt = $pdo->query($sql);
$totale = $stmt->fetchColumn();
echo "Totale studenti: $totale";

// Recupera colonna specifica (0-based)
$sql = "SELECT nome, cognome FROM studenti LIMIT 1";
$stmt = $pdo->query($sql);
$nome = $stmt->fetchColumn(0);    // Prima colonna
$cognome = $stmt->fetchColumn(1); // Seconda colonna

// Utile per liste
$sql = "SELECT nome FROM studenti";
$stmt = $pdo->query($sql);
while ($nome = $stmt->fetchColumn()) {
  echo $nome . "<br>";
}
```

Impostare Fetch Mode Predefinito

```php
<?php
// Metodo 1: Alla connessione
$options = [
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
];
$pdo = new PDO($dsn, $user, $pass, $options);

// Metodo 2: Dopo la connessione
$pdo->setAttribute(
PDO::ATTR_DEFAULT_FETCH_MODE,
PDO::FETCH_ASSOC
);

// Ora fetch() usa FETCH_ASSOC di default
$stmt = $pdo->query("SELECT * FROM studenti");
$studente = $stmt->fetch(); // Automaticamente ASSOC
?>
```

Cos'è SQL Injection?

[!] SQL Injection

Tecnica di attacco che sfrutta input non validati per iniettare codice SQL malevolo in una query.

Conseguenze:

- Furto di dati sensibili
- Modifica/cancellazione dati
- Bypass autenticazione
- Esecuzione comandi sul server
- Compromissione totale del sistema

È uno dei 10 rischi più critici secondo OWASP!

Esempio di Vulnerabilità

× CODICE VULNERABILE - MAI FARE COSÌ!

```php
<?php
// PERICOLOSO! Non usare concatenazione!
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM utenti
WHERE username = '$username'
AND password = '$password'";

$result = $pdo->query($sql);
?>
```

Input malevolo:

Esempio di Vulnerabilità

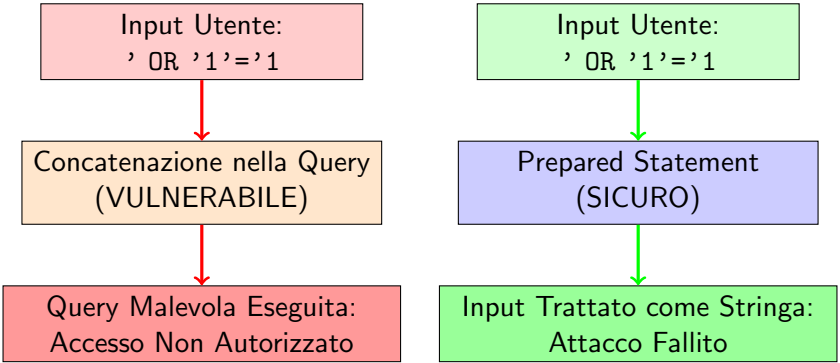
Input malevolo:

```
username: admin' OR '1'='1
password: qualsiasi
```

Query risultante:

```
SELECT * FROM utenti WHERE username = 'admin'
OR '1'='1' AND password = 'qualsiasi'
```

Attacco SQL Injection - Diagramma



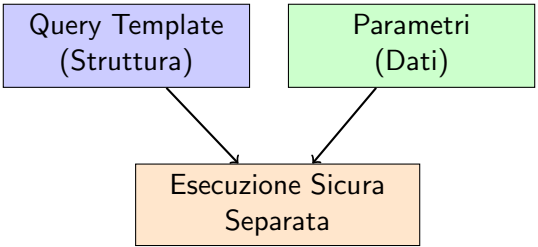
Soluzione: Prepared Statements

✓ CODICE SICURO

```php
<?php
// SICURO! Usa prepared statements!
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM utenti
WHERE username = :username
AND password = :password";

$stmt = $pdo->prepare($sql);
$stmt->execute([
'username' => $username,
'password' => $password
]);

$utente = $stmt->fetch();
```

Come PDO Previene SQL Injection

1. **Separazione**: Query e dati viaggiano separati
2. **Escape automatico**: PDO fa escape di caratteri speciali
3. **Type checking**: Verifica i tipi di dato
4. **Parametrizzazione**: I parametri non sono mai interpretati come SQL



Best Practices di Sicurezza

[*] Regole d'Oro

1. **SEMPRE** usare prepared statements per input utente
2. **MAI** concatenare SQL con variabili
3. **MAI** usare `query()` con dati non fidati
4. Validare input anche prima di PDO
5. Usare `ERRMODE_EXCEPTION`
6. Non mostrare errori SQL all'utente finale
7. Applicare principio del minimo privilegio (utente DB limitato)
8. Hash delle password (mai in chiaro!)

Validazione Input (Difesa Aggiuntiva)

```php
<?php
// Validazione PRIMA di usare PDO
$id = $_GET['id'];

// Verifica che sia un numero
if (!is_numeric($id) || $id < 1) {
  die("ID non valido");
}

// Cast esplicito
$id = (int)$id;

// Ora usalo nel prepared statement
$sql = "SELECT * FROM studenti WHERE id = :id";
$stmt = $pdo->prepare($sql);
$stmt->execute(['id' => $id]);
?>
```

LIKE con Prepared Statements

Problema: Come usare LIKE in modo sicuro?

```php
<?php
$cerca = $_GET['cerca'];

// SBAGLIATO: pattern nella query
// $sql = "SELECT * FROM studenti WHERE nome LIKE '%$cerca%'";

// GIUSTO: pattern nel parametro
$sql = "SELECT * FROM studenti WHERE nome LIKE :pattern";
$stmt = $pdo->prepare($sql);

$pattern = "%{$cerca}%";  // Costruisci pattern in PHP
$stmt->execute(['pattern' => $pattern]);

$risultati = $stmt->fetchAll();
?>

```

Il carattere % va nei DATI, non nella query!

Cos'è una Transazione?

Transazione

Gruppo di operazioni SQL trattate come **un'unica unità atomica**: o si completano tutte con successo, o nessuna ha effetto.

Proprietà ACID:

Atomicity	Tutto o niente
Consistency	Dati consistenti
Isolation	Transazioni indipendenti
Durability	Modifiche permanenti

Esempio: Trasferimento bancario

- Sottrai da conto A
- Aggiungi a conto B
- Se una fallisce, annulla tutto!

Sintassi Base delle Transazioni

```php
<?php
try {
  // 1. INIZIO transazione
  $pdo->beginTransaction();

  // 2. OPERAZIONI multiple
  $stmt1 = $pdo->prepare("UPDATE conti SET saldo = saldo - 100 WHERE id = 1");
  $stmt1->execute();

  $stmt2 = $pdo->prepare("UPDATE conti SET saldo = saldo + 100 WHERE id = 2");
  $stmt2->execute();

  // 3. COMMIT - conferma tutto
  $pdo->commit();
  echo "Trasferimento completato";

} catch (Exception $e) {
```

commit() - Conferma Modifiche

```php
<?php
$pdo->beginTransaction();

// Inserimento multiplo
$sql = "INSERT INTO ordini (cliente, totale)
VALUES (:cliente, :totale)";
$stmt = $pdo->prepare($sql);

$stmt->execute(['cliente' => 'Mario', 'totale' => 150]);
$ordineId = $pdo->lastInsertId();

$sql = "INSERT INTO dettagli_ordine (ordine_id, prodotto)
VALUES (:ordine, :prodotto)";
$stmt = $pdo->prepare($sql);
$stmt->execute(['ordine' => $ordineId, 'prodotto' => 'Laptop']);

// Tutto ok, conferma!
$pdo->commit();
echo "Ordine salvato con successo!";
```

rollBack() - Annulla Modifiche

```php
<?php
try {
  $pdo->beginTransaction();

  // Prima operazione
  $pdo->exec("UPDATE prodotti SET quantita = quantita - 1
  WHERE id = 5");

  // Simula errore
  if (/* condizione errore */) {
    throw new Exception("Stock insufficiente");
  }

  // Seconda operazione
  $pdo->exec("INSERT INTO vendite (prodotto_id) VALUES (5)");

  $pdo->commit();

} catch (Exception $e) {
```

Verifica Stato Transazione

```php
<?php
// Controlla se c'e' una transazione attiva
if ($pdo->inTransaction()) {
  echo "Transazione in corso";
} else {
  echo "Nessuna transazione";
}

// Esempio pratico
try {
  $pdo->beginTransaction();

  // ... operazioni ...

  if ($pdo->inTransaction()) {
    $pdo->commit();
  }

} catch (Exception $e) {
```

Transazioni Annidate (Savepoint)

```php
<?php
try {
  $pdo->beginTransaction();

  // Prima operazione
  $pdo->exec("INSERT INTO log VALUES ('operazione 1')");

  // Savepoint
  $pdo->exec("SAVEPOINT sp1");

  // Seconda operazione
  $pdo->exec("INSERT INTO log VALUES ('operazione 2')");

  // Errore! Torna al savepoint
  $pdo->exec("ROLLBACK TO SAVEPOINT sp1");

  // Commit (salva solo operazione 1)
  $pdo->commit();
```

Quando Usare le Transazioni

Usare transazioni per:

- ✓ Operazioni finanziarie (pagamenti, trasferimenti)
- ✓ Inserimenti correlati su più tabelle
- ✓ Aggiornamenti che devono essere atomici
- ✓ Operazioni con vincoli di integrità complessi
- ✓ Batch di operazioni che devono completare insieme

Non necessarie per:

- × Singole SELECT
- × Operazioni indipendenti
- × Log non critici

Regola: Se un'operazione dipende dal successo di un'altra, usa transazioni!

Try-Catch Completo

```php
<?php
try {
  $pdo = new PDO($dsn, $user, $pass, [
  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
  ]);

  $stmt = $pdo->prepare("SELECT * FROM studenti WHERE id = :id");
  $stmt->execute(['id' => $id]);
  $studente = $stmt->fetch();

  if (!$studente) {
    throw new Exception("Studente non trovato");
  }

} catch (PDOException $e) {
  // Errore PDO specifico
  error_log("Errore DB: " . $e->getMessage());
  die("Errore nel database");

```

Informazioni sugli Errori

```php
<?php
try {
  $stmt = $pdo->prepare("SELECT * FROM tabella_inesistente");
  $stmt->execute();

} catch (PDOException $e) {
  echo "Messaggio: " . $e->getMessage() . "<br>";
  echo "Codice: " . $e->getCode() . "<br>";
  echo "File: " . $e->getFile() . "<br>";
  echo "Linea: " . $e->getLine() . "<br>";

  // SQLSTATE e codice errore MySQL
  echo "SQLSTATE: " . $e->errorInfo[0] . "<br>";
  echo "Codice errore: " . $e->errorInfo[1] . "<br>";
  echo "Messaggio DB: " . $e->errorInfo[2] . "<br>";
}
?>
```

Logging degli Errori

```php
<?php
// Funzione di logging personalizzata
function logError($e) {
  $log = date('Y-m-d H:i:s') . " - ";
  $log .= "SQLSTATE: " . $e->errorInfo[0] . " - ";
  $log .= $e->getMessage() . "\n";

  file_put_contents(
  'logs/db_errors.log',
  $log,
  FILE_APPEND
  );
}

try {
  // ... operazioni DB ...
} catch (PDOException $e) {
  logError($e);
```

Riepilogo Best Practices

1. **Connessione**
   - Usare `ERRMODE_EXCEPTION`
   - Impostare `charset=utf8mb4`
   - Disabilitare emulazione prepared statements
2. **Sicurezza**
   - SEMPRE prepared statements per input utente
   - MAI concatenazione SQL
   - Validare input
   - Non mostrare errori SQL all'utente
3. **Performance**
   - Usare `fetch()` per dataset grandi
   - Riutilizzare prepared statements in loop
   - Chiudere connessioni quando non servono

Checklist Finale

[✓] Checklist PDO

- ☐ Modalità errore su EXCEPTION
- ☐ Charset UTF-8 (utf8mb4)
- ☐ Prepared statements per query parametrizzate
- ☐ Placeholder nominativi (:nome) per chiarezza
- ☐ Gestione errori con try-catch
- ☐ Transazioni per operazioni atomiche
- ☐ Logging errori (non visualizzati all'utente)
- ☐ Validazione input
- ☐ Hash password (password_hash/verify)
- ☐ Principio minimo privilegio per utente DB

Risorse Utili

Documentazione Ufficiale:

- `https://www.php.net/manual/en/book.pdo.php`
- `https://www.php.net/manual/en/pdo.prepared-statements.php`

Sicurezza:

- OWASP - SQL Injection Prevention
- PHP The Right Way - Database

Best Practices:

- PHP-FIG Standards (PSR-12)
- MySQL Performance Best Practices

Esempio Completo - Applicazione CRUD

```php
<?php
class StudenteDAO {
  private $pdo;

  public function __construct($pdo) {
    $this->pdo = $pdo;
  }

  public function inserisci($nome, $cognome, $classe) {
    $sql = "INSERT INTO studenti (nome, cognome, classe)
    VALUES (:nome, :cognome, :classe)";
    $stmt = $this->pdo->prepare($sql);
    $stmt->execute([
    'nome' => $nome,
    'cognome' => $cognome,
    'classe' => $classe
    ]);
    return $this->pdo->lastInsertId();
  }

  public function leggiTutti() {
    $sql = "SELECT * FROM studenti ORDER BY cognome, nome";
    $stmt = $this->pdo->query($sql);
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
  }

```

Esempio Completo - Continuazione

```php
    public function leggiPerId($id) {
      $sql = "SELECT * FROM studenti WHERE id = :id";
      $stmt = $this->pdo->prepare($sql);
      $stmt->execute(['id' => $id]);
      return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    public function aggiorna($id, $nome, $cognome, $classe) {
      $sql = "UPDATE studenti
      SET nome = :nome, cognome = :cognome, classe = :classe
      WHERE id = :id";
      $stmt = $this->pdo->prepare($sql);
      return $stmt->execute([
      'id' => $id,
      'nome' => $nome,
      'cognome' => $cognome,
      'classe' => $classe
      ]);
    }

    public function elimina($id) {
      $sql = "DELETE FROM studenti WHERE id = :id";
      $stmt = $this->pdo->prepare($sql);
      return $stmt->execute(['id' => $id]);
    }
  }
  ?>
```

PDO: La Scelta Giusta per PHP Moderno

Abbiamo visto:

- Connessione sicura e configurata
- Prepared statements per sicurezza
- Metodi fetch per recupero dati
- Transazioni per integrità
- Prevenzione SQL Injection
- Best practices di sviluppo

Grazie per l'attenzione!

Domande?

massimo.pippi@ferrmi-ceci.edu.it