

# PDO: PHP Data Objects

## Accesso sicuro ai database MySQL

Prof. Massimo Fedeli

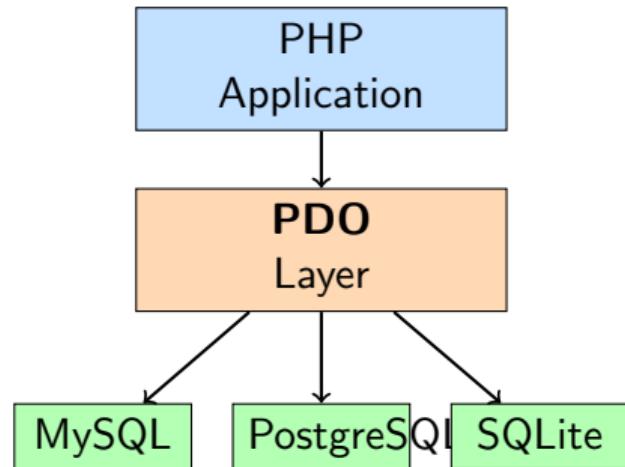
IIS Fermi Sacconi Ceci - Ascoli Piceno

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**



## ✓ 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

```
1  <?php
2  // Parametri di connessione
3  $host = 'localhost';
4  $dbname = 'scuola';
5  $username = 'root';
6  $password = '';
7
8  try {
9      // Creazione oggetto PDO
10     $pdo = new PDO(
11         "mysql:host=$host;dbname=$dbname;charset=utf8mb4",
12         $username,
13         $password
14     );
15
16     echo "Connessione riuscita!";
17
18 } catch (PDOException $e) {
19     die("Errore connessione: " . $e->getMessage());
```

## 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

```
1  <?php
2  $options = [
3  // Modalita' errore: eccezioni
4  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
5
6  // Fetch mode predefinito
7  PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
8
9  // Disabilita emulazione prepared statements
10 PDO::ATTR_EMULATE_PREPARES => false,
11
12 // Connessione persistente (opzionale)
13 PDO::ATTR_PERSISTENT => false
14 ];
15
16 $pdo = new PDO($dsn, $username, $password, $options);
17 ?>
18
```

## 1. ERMODE\_SILENT

```
1 // Nessun errore mostrato
2 // Controllo manuale
3 if (!$stmt->execute()) {
4     print_r($stmt->errorInfo());
5 }
6
```

## 2. ERMODE\_WARNING

```
1 // Warning PHP standard
2 // Non interrompe esecuzione
3
```

## 3. ERMODE\_EXCEPTION ✓

```
1 try {
2     $stmt = $pdo->query($sql);
3 } catch (PDOException $e) {
4     echo "Errore: ";
5     echo $e->getMessage();
6 }
7
```

**Best Practice:** Usare sempre EXCEPTION

# Chiusura Connessione

```
1  <?php
2  // Connessione aperta
3  $pdo = new PDO($dsn, $username, $password);
4
5  // ... operazioni sul database ...
6
7  // Chiusura esplicita
8  $pdo = null;
9
10 // La connessione si chiude automaticamente
11 // alla fine dello script
12 ?>
```

[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

```
1  <?php
2      $sql = "SELECT * FROM studenti";
3
4      // Esecuzione diretta
5      $stmt = $pdo->query($sql);
6
7      // $stmt e' un oggetto PDOStatement
8      echo "Righe trovate: " . $stmt->rowCount();
9
10 ?>
```

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

# exec() per INSERT, UPDATE, DELETE

```
1  <?php
2  // exec() restituisce il numero di righe affette
3  $sql = "DELETE FROM studenti WHERE voto < 6";
4  $righe = $pdo->exec($sql);
5
6  echo "Eliminate $righe righe";
7
8  // Altro esempio
9  $sql = "UPDATE studenti SET classe = '5A'
10 WHERE classe = '4A'";
11 $righe = $pdo->exec($sql);
12
13 echo "Aggiornate $righe righe";
14 ?>
15
```

## Differenza:

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



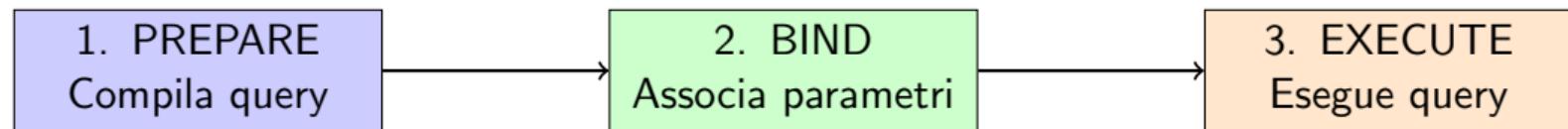
# Recupero Ultimo ID Inserito

```
1  <?php
2   $sql = "INSERT INTO studenti (nome, cognome, classe)
3     VALUES ('Mario', 'Rossi', '3A')";
4
5   $pdo->exec($sql);
6
7   // Ottieni l'ID auto-incrementato
8   $ultimoId = $pdo->lastInsertId();
9
10  echo "Nuovo studente inserito con ID: $ultimoId";
11
12 ?>
```

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:

```
1  <?php
2 // 1. PREPARE - con placeholder ?
3 $sql = "SELECT * FROM studenti WHERE classe = ?";
4 $stmt = $pdo->prepare($sql);
5
6 // 2. EXECUTE - passa parametri
7 $stmt->execute(['3A']);
8
9 // 3. FETCH - recupera risultati
10 $risultati = $stmt->fetchAll();
11
12 foreach ($risultati as $studente) {
13     echo $studente['nome'] . " " . $studente['cognome'];
14 }
15
16 ?>
```

# Placeholder Posizionali (?)

```
1  <?php
2   $sql = "SELECT * FROM studenti
3   WHERE classe = ? AND voto >= ?";
4
5   $stmt = $pdo->prepare($sql);
6
7   // L'ordine dei parametri DEVE corrispondere
8   $stmt->execute(['3A', 7]);
9
10  // Esempio con INSERT
11  $sql = "INSERT INTO studenti (nome, cognome, classe)
12  VALUES (?, ?, ?)";
13  $stmt = $pdo->prepare($sql);
14  $stmt->execute(['Mario', 'Rossi', '3A']);
15
16 ?>
```

**Pro:** Più compatti

**Contro:** Ordine critico, meno leggibile

# Placeholder Nominativi (:nome)

```
1  <?php
2   $sql = "SELECT * FROM studenti
3   WHERE classe = :classe AND voto >= :voto_min";
4
5   $stmt = $pdo->prepare($sql);
6
7   // Array associativo - ordine non importante!
8   $stmt->execute([
9     ':classe' => '3A',
10    ':voto_min' => 7
11  ]);
12
13  // Oppure senza i due punti nelle chiavi
14  $stmt->execute([
15    'classe' => '3A',
16    'voto_min' => 7
17  ]);
18
19 ?>
```

## execute() con Array

```
1  <?php
2  // INSERT multipli
3  $sql = "INSERT INTO studenti (nome, cognome, classe)
4  VALUES (:nome, :cognome, :classe)";
5  $stmt = $pdo->prepare($sql);
6
7  $studenti = [
8      ['nome' => 'Mario', 'cognome' => 'Rossi', 'classe' => '3A'],
9      ['nome' => 'Laura', 'cognome' => 'Bianchi', 'classe' => '3A'],
10     ['nome' => 'Giuseppe', 'cognome' => 'Verdi', 'classe' => '3B']
11 ];
12
13  foreach ($studenti as $studente) {
14      $stmt->execute($studente);
15  }
16
17  echo "Inseriti " . count($studenti) . " studenti";
18
19 ?>
```

# bindValue() - Base

## Associa un valore a un parametro:

```
1 <?php
2     $sql = "SELECT * FROM studenti WHERE classe = :classe";
3     $stmt = $pdo->prepare($sql);
4
5     // Binding singolo
6     $stmt->bindValue(':classe', '3A');
7     $stmt->execute();
8
9     // Equivalente a:
10    // $stmt->execute([':classe' => '3A']);
11    ?>
12
```

Utile quando:

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

# bindParam() vs bindValue()

## bindValue() - per valore

```
1  $classe = '3A';
2  $stmt->bindValue(':c', $classe);
3  $classe = '3B'; // Non influisce
4  $stmt->execute();
5  // Query con '3A'
6
```

## bindParam() - per riferimento

```
1  $classe = '3A';
2  $stmt->bindParam(':c', $classe);
3  $classe = '3B'; // Cambia!
4  $stmt->execute();
5  // Query con '3B'
6
```

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

# bindParam() - Esempio Pratico

```
1  <?php
2  $sql = "INSERT INTO log (utente, azione, timestamp)
3    VALUES (:utente, :azione, :ts)";
4  $stmt = $pdo->prepare($sql);
5
6  // Binding per riferimento
7  $stmt->bindParam(':utente', $utente);
8  $stmt->bindParam(':azione', $azione);
9  $stmt->bindParam(':ts', $timestamp);
10
11 // Esecuzione multipla con valori diversi
12 $utente = 'mario'; $azione = 'login';
13 $timestamp = time();
14 $stmt->execute();
15
16 $utente = 'laura'; $azione = 'logout';
17 $timestamp = time();
18 $stmt->execute();
19 ?>
```

# Tipi di Dati (PDO::PARAM\_\*)

```
1  <?php
2  $sql = "SELECT * FROM studenti
3  WHERE voto >= :voto AND attivo = :attivo";
4  $stmt = $pdo->prepare($sql);
5
6  // Specificare il tipo esplicitamente
7  $stmt->bindValue(':voto', 6, PDO::PARAM_INT);
8  $stmt->bindValue(':attivo', true, PDO::PARAM_BOOL);
9
10 $stmt->execute();
11 ?>
12
```

## Tipi disponibili:

- PDO::PARAM\_INT - Intero
- PDO::PARAM\_BOOL - Booleano
- PDO::PARAM\_STR - Stringa (default)
- PDO::PARAM\_NULL - NULL

# fetch() - Recupero Riga Singola

```
1  <?php
2  $sql = "SELECT * FROM studenti WHERE id = :id";
3  $stmt = $pdo->prepare($sql);
4  $stmt->execute(['id' => 1]);
5
6  // Recupera UNA riga
7  $studente = $stmt->fetch();
8
9  if ($studente) {
10    echo $studente['nome'] . " " . $studente['cognome'];
11  } else {
12    echo "Studente non trovato";
13  }
14
15 ?>
```

## Comportamento:

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

# fetch() in Loop

```
1 <?php
2     $sql = "SELECT nome, cognome, voto FROM studenti
3         WHERE classe = '3A'";
4     $stmt = $pdo->query($sql);
5
6     // Ciclo su tutte le righe
7     while ($riga = $stmt->fetch()) {
8         echo $riga['nome'] . " " . $riga['cognome'];
9         echo " - Voto: " . $riga['voto'] . "<br>";
10    }
11
12 ?>
```

**Vantaggio:** Consuma poca memoria anche con molti risultati

**Quando usarlo:** Elaborazione riga per riga, grandi dataset

# fetchAll() - Tutte le Righe

```
1  <?php
2   $sql = "SELECT * FROM studenti WHERE classe = '3A'";
3   $stmt = $pdo->query($sql);
4
5   // Recupera TUTTE le righe in un array
6   $studenti = $stmt->fetchAll();
7
8   echo "Trovati " . count($studenti) . " studenti<br>";
9
10  foreach ($studenti as $studente) {
11      echo $studente['nome'] . "<br>";
12  }
13  ?>
14
```

## Quando usarlo:

- Risultati piccoli/medi
- Serve contare le righe prima di elaborarle

# 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

```
1  <?php
2   $sql = "SELECT * FROM studenti WHERE id = 1";
3   $stmt = $pdo->query($sql);
4
5   // Recupera come oggetto stdClass
6   $studente = $stmt->fetch(PDO::FETCH_OBJ);
7
8   echo $studente->nome;           // Propriet , non chiavi!
9   echo $studente->cognome;
10  echo $studente->classe;
11  ?>
12
```

Utile per:

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

# FETCH\_CLASS - Oggetto Personalizzato

```
1  <?php
2  class Studente {
3      public $id;
4      public $nome;
5      public $cognome;
6
7      public function nomeCompleto() {
8          return $this->nome . " " . $this->cognome;
9      }
10 }
11
12 $sql = "SELECT * FROM studenti WHERE classe = '3A'";
13 $stmt = $pdo->query($sql);
14
15 // Popola oggetti della classe Studente
16 $studenti = $stmt->fetchAll(PDO::FETCH_CLASS, 'Studente');
17
18 foreach ($studenti as $s) {
19     echo $s->nomeCompleto(); // Usa metodo della classe!
```

# fetchColumn() - Singola Colonna

```
1  <?php
2 // Recupera solo una colonna
3 $sql = "SELECT COUNT(*) FROM studenti";
4 $stmt = $pdo->query($sql);
5 $totale = $stmt->fetchColumn();
6 echo "Totale studenti: $totale";
7
8 // Recupera colonna specifica (0-based)
9 $sql = "SELECT nome, cognome FROM studenti LIMIT 1";
10 $stmt = $pdo->query($sql);
11 $nome = $stmt->fetchColumn(0);      // Prima colonna
12 $cognome = $stmt->fetchColumn(1); // Seconda colonna
13
14 // Utile per liste
15 $sql = "SELECT nome FROM studenti";
16 $stmt = $pdo->query($sql);
17 while ($nome = $stmt->fetchColumn()) {
18     echo $nome . "<br>";
19 }
```

# Impostare Fetch Mode Predefinito

```
1  <?php
2 // Metodo 1: Alla connessione
3 $options = [
4 PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
5 ];
6 $pdo = new PDO($dsn, $user, $pass, $options);
7
8 // Metodo 2: Dopo la connessione
9 $pdo->setAttribute(
10 PDO::ATTR_DEFAULT_FETCH_MODE,
11 PDO::FETCH_ASSOC
12 );
13
14 // Ora fetch() usa FETCH_ASSOC di default
15 $stmt = $pdo->query("SELECT * FROM studenti");
16 $studente = $stmt->fetch(); // Automaticamente ASSOC
17 ?>
18
```

# 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!

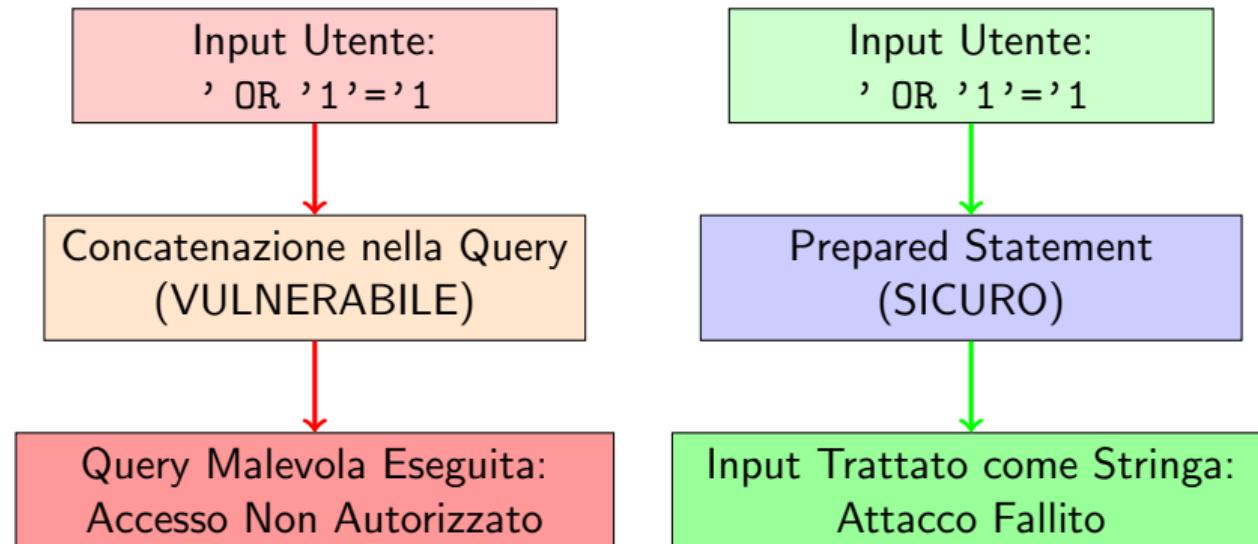
# Esempio di Vulnerabilità

✗ CODICE VULNERABILE - MAI FARE COSÌ!

```
1      <?php
2      // PERICOLOSO! Non usare concatenazione!
3      $username = $_POST['username'];
4      $password = $_POST['password'];
5
6      $sql = "SELECT * FROM utenti
7      WHERE username = '$username'
8      AND password = '$password'";
9
10     $result = $pdo->query($sql);
11
12     ?>
```

**Input malevolo:**

# Attacco SQL Injection - Diagramma



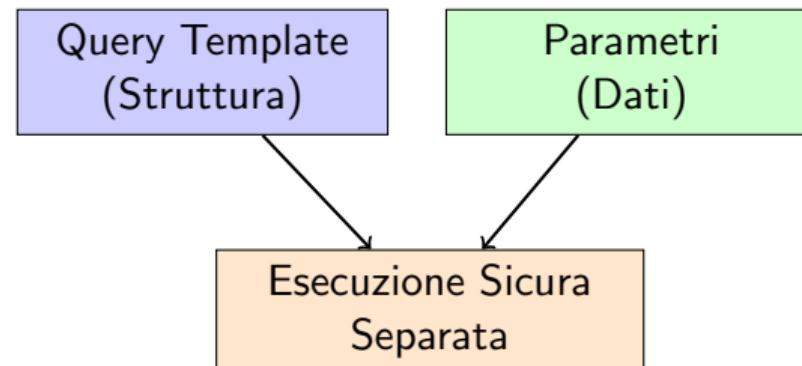
# Soluzione: Prepared Statements

## ✓ CODICE SICURO

```
1      <?php
2      // SICURO! Usa prepared statements!
3      $username = $_POST['username'];
4      $password = $_POST['password'];
5
6      $sql = "SELECT * FROM utenti
7          WHERE username = :username
8          AND password = :password";
9
10     $stmt = $pdo->prepare($sql);
11     $stmt->execute([
12         'username' => $username,
13         'password' => $password
14     ]);
15
16     $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



## [\*] 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)

```
1  <?php
2  // Validazione PRIMA di usare PDO
3  $id = $_GET['id'];
4
5  // Verifica che sia un numero
6  if (!is_numeric($id) || $id < 1) {
7      die("ID non valido");
8  }
9
10 // Cast esplicito
11 $id = (int)$id;
12
13 // Ora usalo nel prepared statement
14 $sql = "SELECT * FROM studenti WHERE id = :id";
15 $stmt = $pdo->prepare($sql);
16 $stmt->execute(['id' => $id]);
17 ?>
18
```

# LIKE con Prepared Statements

**Problema:** Come usare LIKE in modo sicuro?

```
1 <?php
2     $cerca = $_GET['cerca'];
3
4     // SBAGLIATO: pattern nella query
5     // $sql = "SELECT * FROM studenti WHERE nome LIKE '%$cerca%'";
6
7     // GIUSTO: pattern nel parametro
8     $sql = "SELECT * FROM studenti WHERE nome LIKE :pattern";
9     $stmt = $pdo->prepare($sql);
10
11    $pattern = "%{$cerca}%"; // Costruisce pattern in PHP
12    $stmt->execute(['pattern' => $pattern]);
13
14    $risultati = $stmt->fetchAll();
15
16    ?>
```

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

```
1  <?php
2  try {
3      // 1. INIZIO transazione
4      $pdo->beginTransaction();
5
6      // 2. OPERAZIONI multiple
7      $stmt1 = $pdo->prepare("UPDATE conti SET saldo = saldo - 100 WHERE
id = 1");
8      $stmt1->execute();
9
10     $stmt2 = $pdo->prepare("UPDATE conti SET saldo = saldo + 100 WHERE
id = 2");
11     $stmt2->execute();
12
13     // 3. COMMIT - conferma tutto
14     $pdo->commit();
15     echo "Trasferimento completato";
16
17 } catch (Exception $e) {
```

# commit() - Conferma Modifiche

```
1  <?php
2      $pdo->beginTransaction();
3
4      // Inserimento multiplo
5      $sql = "INSERT INTO ordini (cliente, totale)
6          VALUES (:cliente, :totale)";
7      $stmt = $pdo->prepare($sql);
8
9      $stmt->execute(['cliente' => 'Mario', 'totale' => 150]);
10     $ordineId = $pdo->lastInsertId();
11
12     $sql = "INSERT INTO dettagli_ordine (ordine_id, prodotto)
13         VALUES (:ordine, :prodotto)";
14     $stmt = $pdo->prepare($sql);
15     $stmt->execute(['ordine' => $ordineId, 'prodotto' => 'Laptop']);
16
17     // Tutto ok, conferma!
18     $pdo->commit();
19     echo "Ordine salvato con successo!";
```

## rollBack() - Annulla Modifiche

```
1  <?php
2  try {
3      $pdo->beginTransaction();
4
5      // Prima operazione
6      $pdo->exec("UPDATE prodotti SET quantita = quantita - 1
7      WHERE id = 5");
8
9      // Simula errore
10     if /* condizione errore */) {
11         throw new Exception("Stock insufficiente");
12     }
13
14     // Seconda operazione
15     $pdo->exec("INSERT INTO vendite (prodotto_id) VALUES (5)");
16
17     $pdo->commit();
18
19 } catch (Exception $e) {
```

# Verifica Stato Transazione

```
1  <?php
2  // Controlla se c'e' una transazione attiva
3  if ($pdo->inTransaction()) {
4      echo "Transazione in corso";
5  } else {
6      echo "Nessuna transazione";
7  }
8
9  // Esempio pratico
10 try {
11     $pdo->beginTransaction();
12
13     // ... operazioni ...
14
15     if ($pdo->inTransaction()) {
16         $pdo->commit();
17     }
18
19 } catch (Exception $e) {
```

# Transazioni Annidate (Savepoint)

```
1  <?php
2  try {
3      $pdo->beginTransaction();
4
5      // Prima operazione
6      $pdo->exec("INSERT INTO log VALUES ('operazione 1')");
7
8      // Savepoint
9      $pdo->exec("SAVEPOINT sp1");
10
11     // Seconda operazione
12     $pdo->exec("INSERT INTO log VALUES ('operazione 2')");
13
14     // Errore! Torna al savepoint
15     $pdo->exec("ROLLBACK TO SAVEPOINT sp1");
16
17     // Commit (salva solo operazione 1)
18     $pdo->commit();
19
```

# 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

```
1  <?php
2  try {
3      $pdo = new PDO($dsn, $user, $pass, [
4          PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
5      ]);
6
7      $stmt = $pdo->prepare("SELECT * FROM studenti WHERE id = :id");
8      $stmt->execute(['id' => $id]);
9      $studente = $stmt->fetch();
10
11     if (!$studente) {
12         throw new Exception("Studente non trovato");
13     }
14
15 } catch (PDOException $e) {
16     // Errore PDO specifico
17     error_log("Errore DB: " . $e->getMessage());
18     die("Errore nel database");
19 }
```

# Informazioni sugli Errori

```
1  <?php
2  try {
3      $stmt = $pdo->prepare("SELECT * FROM tabella_inesistente");
4      $stmt->execute();
5
6  } catch (PDOException $e) {
7      echo "Messaggio: " . $e->getMessage() . "<br>";
8      echo "Codice: " . $e->getCode() . "<br>";
9      echo "File: " . $e->getFile() . "<br>";
10     echo "Linea: " . $e->getLine() . "<br>";
11
12     // SQLSTATE e codice errore MySQL
13     echo "SQLSTATE: " . $e->errorInfo[0] . "<br>";
14     echo "Codice errore: " . $e->errorInfo[1] . "<br>";
15     echo "Messaggio DB: " . $e->errorInfo[2] . "<br>";
16 }
17 ?>
```

# Logging degli Errori

```
1  <?php
2  // Funzione di logging personalizzata
3  function logError($e) {
4      $log = date('Y-m-d H:i:s') . " - ";
5      $log .= "SQLSTATE: " . $e->errorInfo[0] . " - ";
6      $log .= $e->getMessage() . "\n";
7
8      file_put_contents(
9          'logs/db_errors.log',
10         $log,
11         FILE_APPEND
12     );
13 }
14
15 try {
16     // ... operazioni DB ...
17 } catch (PDOException $e) {
18     logError($e);
19 }
```

# 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 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

## 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

```
1  <?php
2  class StudenteDAO {
3      private $pdo;
4
5      public function __construct($pdo) {
6          $this->pdo = $pdo;
7      }
8
9      public function inserisci($nome, $cognome, $classe) {
10         $sql = "INSERT INTO studenti (nome, cognome, classe)
11             VALUES (:nome, :cognome, :classe)";
12         $stmt = $this->pdo->prepare($sql);
13         $stmt->execute([
14             'nome' => $nome,
15             'cognome' => $cognome,
16             'classe' => $classe
17         ]);
18         return $this->pdo->lastInsertId();
19     }
20
21     public function leggiTutti() {
22         $sql = "SELECT * FROM studenti ORDER BY cognome, nome";
23         $stmt = $this->pdo->query($sql);
24         return $stmt->fetchAll(PDO::FETCH_ASSOC);
25     }
26 }
```

# Esempio Completo - Continuazione

```
1  public function leggiPerId($id) {
2      $sql = "SELECT * FROM studenti WHERE id = :id";
3      $stmt = $this->pdo->prepare($sql);
4      $stmt->execute(['id' => $id]);
5      return $stmt->fetch(PDO::FETCH_ASSOC);
6  }
7
8  public function aggiorna($id, $nome, $cognome, $classe) {
9      $sql = "UPDATE studenti
10         SET nome = :nome, cognome = :cognome, classe = :classe
11        WHERE id = :id";
12      $stmt = $this->pdo->prepare($sql);
13      return $stmt->execute([
14          'id' => $id,
15          'nome' => $nome,
16          'cognome' => $cognome,
17          'classe' => $classe
18      ]);
19  }
20
21  public function elimina($id) {
22      $sql = "DELETE FROM studenti WHERE id = :id";
23      $stmt = $this->pdo->prepare($sql);
24      return $stmt->execute(['id' => $id]);
25  }
26
27 ?>
```

## 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](mailto:massimo.pippi@ferrmi-ceci.edu.it)