# RIASSUNTONE

**✅ Connessione al Database (OO)**

Per stabilire una connessione al database si usa:

php

CopiaModifica

$conn = new mysqli('localhost', 'username', 'password', 'DB\_name');

**Controllo degli errori di connessione**

È buona pratica verificare se ci sono errori durante la connessione:

php

CopiaModifica

if ($conn->connect\_error) {

die('Errore di connessione (' . $conn->connect\_errno . ') ' . $conn->connect\_error);

} else {

echo 'Connesso. ' . $conn->host\_info . "\n";

}

**Esempio di errore**:  
Errore di connessione (1049) Unknown database 'es1'  
Questo significa che il database specificato non esiste.

**Chiusura della connessione**

Dopo aver terminato le operazioni:

php

CopiaModifica

$conn->close();

**🛠️ Esecuzione di Query (OO)**

Esempio di creazione di un database:

php

CopiaModifica

if (!$conn->query("CREATE DATABASE biblioteca")) {

die($conn->error);

}

Se la creazione va a buon fine, si seleziona il database:

php

CopiaModifica

$conn->query("USE biblioteca");

**🔢 Funzioni utili di MySQLi**

| **Funzione** | **Descrizione** |
| --- | --- |
| mysqli\_num\_rows($risultato) | Restituisce il numero di righe ritornate da una query SELECT |
| mysqli\_affected\_rows($conn) | Restituisce quante righe sono state modificate (INSERT, UPDATE, DELETE) |

**📋 Lettura dei risultati da una SELECT (OO)**

Esempio:

php

CopiaModifica

$query = "SELECT nome, cognome FROM person ORDER BY ID";

$result = $conn->query($query);

**1. Array numerico**

php

CopiaModifica

while ($row = $result->fetch\_array(MYSQLI\_NUM)) {

echo $row[0] . " " . $row[1] . "<br>";

}

**2. Array associativo**

php

CopiaModifica

while ($row = $result->fetch\_array(MYSQLI\_ASSOC)) {

echo $row['nome'] . " " . $row['cognome'] . "<br>";

}

**3. Array misto (numerico + associativo)**

php

CopiaModifica

while ($row = $result->fetch\_array(MYSQLI\_BOTH)) {

echo $row[0] . " " . $row['cognome'] . "<br>";

}

**4. Libera la memoria dei risultati**

php

CopiaModifica

$result->free();

MySQLi in modalità object-oriented fornisce un'interfaccia più sicura e moderna per lavorare con database MySQL rispetto alla vecchia estensione mysql\_. Le principali funzionalità includono:

* Connessione gestita tramite oggetti
* Gestione avanzata degli errori
* Vari modi per accedere ai risultati (numerico, associativo, entrambi)
* Funzioni per monitorare gli effetti delle query

**🔍 Perché usare PDO?**

PHP supporta vari database (MySQL, PostgreSQL, SQLite, ecc.), ma:

* Ogni database ha un **insieme di funzioni diverso** (mysql\_query(), pg\_query(), SQLite3::query(), ecc.)
* Questo porta a **mancanza di portabilità** tra DB
* Le interfacce native sono **incoerenti e verbose**
* PDO fornisce un'interfaccia **unificata, orientata agli oggetti** per accedere a diversi DB
* È **disponibile da PHP 5.1**

**⚠️ PDO non è un Database Abstraction Layer:**  
Non riscrive SQL e non simula funzioni assenti nei DB.

**✅ Caratteristiche di PDO**

* Interfaccia uniforme per diversi database
* Richiede driver specifici (es. pdo\_mysql, pdo\_pgsql, pdo\_sqlite)
* Offre **supporto nativo ai prepared statements**
* Migliore gestione degli errori (eccezioni)
* Maggiore sicurezza (contro SQL Injection)

**⚙️ Installazione di PDO**

* **Linux/Unix**: Compilare con supporto a PDO
* **Windows**: Abilitare la DLL nel php.ini (es. extension=php\_pdo\_mysql.dll)

**🔌 Connessione a un Database**

php

CopiaModifica

try {

$dbh = new PDO('mysql:host=localhost;dbname=db;port=3306', 'user', 'pass');

$dbh->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

} catch (PDOException $e) {

echo $e->getMessage();

}

Altri esempi:

php

CopiaModifica

$file\_db = new PDO('sqlite:test.lite');

**Chiusura connessione:**

php

CopiaModifica

$dbh = null;

$file\_db = null;

**📄 Esecuzione di Query**

**🔹 Query diretta + ciclo foreach**

php

CopiaModifica

$sth = $dbh->query('SELECT \* FROM test');

foreach ($sth as $row) {

echo $row['titolo'] . '<br>';

}

**🔹 Query con fetch()**

php

CopiaModifica

$stmt = $dbh->query('SELECT nome, cognome FROM girl');

while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)) {

echo $row['nome'] . ' ' . $row['cognome'];

}

**🧠 Prepared Statements (Query Precompilate)**

**🔹 Sintassi base**

php

CopiaModifica

$sth = $dbh->prepare('SELECT \* FROM test');

$sth->execute();

while ($row = $sth->fetch()) {

print\_r($row);

}

**✍️ Esempi di INSERT con Prepared Statements**

**1. Uso di bindParam**

php

CopiaModifica

$sth = $dbh->prepare("INSERT INTO REGISTRY (id, value) VALUES (:id, :value)");

$sth->bindParam(':id', $id, PDO::PARAM\_INT);

$sth->bindParam(':value', $value, PDO::PARAM\_STR, 12);

$id = 1;

$value = 'one';

$sth->execute();

$id = 2;

$value = 'two';

$sth->execute();

**2. Uso con array (più compatto)**

php

CopiaModifica

$sth = $dbh->prepare("INSERT INTO REGISTRY (id, value) VALUES (:id, :value)");

$sth->execute(['id' => 1, 'value' => 'one']);

**🧮 bindParam() vs bindValue()**

**🔸 bindParam() → lega la variabile (valore preso al momento dell’esecuzione)**

php

CopiaModifica

$sex = 'male';

$s = $dbh->prepare('SELECT name FROM students WHERE sex = :sex');

$s->bindParam(':sex', $sex);

$sex = 'female';

$s->execute(); // usa 'female'

**🔸 bindValue() → lega il valore (valore preso subito)**

php

CopiaModifica

$sex = 'male';

$s = $dbh->prepare('SELECT name FROM students WHERE sex = :sex');

$s->bindValue(':sex', $sex);

$sex = 'female';

$s->execute(); // usa 'male'

**🎯 Positional Parameters (senza nomi)**

php

CopiaModifica

$calories = 150;

$colour = 'red';

$sth = $dbh->prepare('SELECT name, colour FROM fruit WHERE calories < ? AND colour = ?');

$sth->execute([$calories, $colour]);

**📋 Uso di fetchAll()**

php

CopiaModifica

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

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

$stmt = $pdo->prepare('SELECT \* FROM mytable WHERE active = 1 ORDER BY name ASC');

$stmt->setFetchMode(PDO::FETCH\_ASSOC);

$stmt->execute();

$rows = $stmt->fetchAll();

$num\_rows = count($rows);

foreach ($rows as $r) {

// primo ciclo

}

foreach ($rows as $r) {

// secondo ciclo

}

🔹 **Nota**: fetchAll() consuma più memoria. Se ti serve solo un passaggio, usa fetch().

**PDO** è la soluzione ideale se:

* Vuoi un codice più **portabile** tra diversi DB
* Cerchi **sicurezza** contro SQL injection
* Preferisci uno **stile moderno e orientato agli oggetti**
* Hai bisogno di gestire **query dinamiche** con parametri

**🔐 Sicurezza: OWASP Top 10 e Mitigazioni**

**1. Broken Access Control**

* **Vulnerabilità**: accesso non autorizzato a risorse tramite URL modificato (id=13).
* **Mitigazioni**:
  + Verificare sempre i permessi lato server.
  + Non usare direttamente gli ID nei parametri URL.
  + Validare rigorosamente gli input utente.

**2. Identification and Authentication Failures**

* **Vulnerabilità**:
  + Archiviazione di password in chiaro o con MD5/SHA1.
* **Mitigazioni**:
  + Usare algoritmi sicuri (es. bcrypt, argon2).
  + Imporre politiche di password forti.
  + Limitare i tentativi di accesso.
  + Implementare 2FA.

**3. Sensitive Data Exposure**

* **Vulnerabilità**: trasmissione o archiviazione di dati sensibili in chiaro (es. HTTP o backup non cifrati).
* **Mitigazioni**:
  + Usare HTTPS.
  + Non usare algoritmi deboli per cifrare i dati.
  + Crittografare i dati a riposo e in transito.

**4. Security Misconfiguration**

* **Vulnerabilità**: errori visibili, variabili non inizializzate, file esposti.
* **Mitigazioni**:
  + Inizializzare le variabili di sicurezza.
  + Disattivare la visualizzazione di errori in produzione.
  + Configurare correttamente il server e l’applicazione.

**5. Directory Traversal**

* **Vulnerabilità**: accesso a file esterni tramite manipolazione dell’input (../etc/passwd).
* **Mitigazioni**:
  + Validare/sanitizzare i nomi file ricevuti dall’utente.
  + Limitare l’accesso con .htaccess o configurazione server.

**6. Cross-Site Request Forgery (CSRF)**

* **Vulnerabilità**: azioni non autorizzate da siti terzi (es. trasferimenti bancari).
* **Mitigazioni**:
  + Usare solo metodi POST per operazioni sensibili.
  + Includere un CSRF token univoco nei form.

**7. Cross-Site Scripting (XSS)**

* **Vulnerabilità**: esecuzione di script malevoli nel browser dell’utente.
* **Mitigazioni**:
  + Sanitizzare l’input (es. htmlspecialchars()).
  + Implementare una Content Security Policy (CSP).

**8. SQL Injection**

* **Vulnerabilità**: esecuzione arbitraria di comandi SQL tramite input utente.
* **Mitigazioni**:
  + Usare **Prepared Statements** (PDO/MySQLi).
  + Evitare concatenazioni dirette nei comandi SQL.
  + In alternativa, usare mysqli\_real\_escape\_string() (solo MySQLi) o addslashes() (meno sicuro).

**🧱 Uso Sicuro del Database: PDO vs MySQLi**

**🔷 PDO (PHP Data Objects)**

* **Caratteristiche**:
  + Unifica l’accesso a diversi DBMS.
  + Supporta Prepared Statements.
  + Migliora sicurezza e leggibilità del codice.
  + Richiede driver (es. pdo\_mysql).

**✔️ Connessione**

php

CopiaModifica

$dbh = new PDO('mysql:host=localhost;dbname=test', 'user', 'pass');

$dbh->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

**📌 Prepared Statements (con nomi o posizionali)**

php

CopiaModifica

$stmt = $dbh->prepare("INSERT INTO utenti (id, email) VALUES (:id, :email)");

$stmt->bindParam(':id', $id, PDO::PARAM\_INT);

$stmt->bindParam(':email', $email, PDO::PARAM\_STR);

Oppure:

php

CopiaModifica

$stmt = $dbh->prepare("INSERT INTO utenti VALUES (?, ?)");

$stmt->execute([$id, $email]);

**🔎 Fetch e fetchAll()**

php

CopiaModifica

$stmt = $dbh->query("SELECT \* FROM utenti");

while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)) {

echo $row['email'];

}

* fetchAll() è utile ma consuma più memoria.

**🔶 MySQLi**

* Più specifico per MySQL, ma simile a PDO per quanto riguarda Prepared Statements.

**🧪 Esempio con bind\_param**

php

CopiaModifica

$stmt = $conn->prepare("INSERT INTO menu (nome, prezzo) VALUES (?, ?)");

$stmt->bind\_param("sd", $nome, $prezzo); // s: string, d: double

$nome = "Pizza"; $prezzo = 7.5;

$stmt->execute();

**🕵️‍♂️ Estrazione dati**

* bind\_result():

php

CopiaModifica

$stmt->bind\_result($nome, $prezzo);

while ($stmt->fetch()) {

echo "$nome: $prezzo";

}

* get\_result() (richiede MySQL Native Driver):

php

CopiaModifica

$result = $stmt->get\_result();

while ($row = $result->fetch\_assoc()) {

echo $row['nome'];

}

**📌 Differenza: bindParam() vs bindValue() in PDO**

| **Metodo** | **Comportamento** |
| --- | --- |
| bindParam() | Lega **la variabile**, valore preso al momento dell’esecuzione |
| bindValue() | Lega **il valore attuale**, preso subito |

**✅ Conclusioni**

* L’uso di **Prepared Statements** (sia con PDO che MySQLi) è la difesa principale contro le SQL Injection.
* La **sicurezza applicativa** non si limita al codice SQL: riguarda anche la **gestione dei dati sensibili**, **autenticazione**, **configurazioni**, **accessi**, e **interfacce utente**.
* Usare framework e librerie moderne aiuta a ridurre i rischi (es. ORM, sistemi di autenticazione integrati, middleware CSRF/XSS,…

**🔹 Cos’è ORM (Object-Relational Mapping)**

L’**ORM** è una tecnica che mappa le **entità di un database relazionale** su **oggetti** nel linguaggio di programmazione. Permette agli sviluppatori di **interagire con il DB usando codice orientato agli oggetti**, senza scrivere direttamente SQL.

**✅ Vantaggi:**

* Astrazione e semplificazione dell'accesso ai dati
* Portabilità tra DBMS diversi
* Riduzione del codice boilerplate
* Gestione automatica della concorrenza
* Modularizzazione dell’architettura (separazione tra logica e accesso dati)

**🔹 Tipologie di ORM**

**🔸 Active Record**

* Ogni entità ha una classe corrispondente.
* L’oggetto è responsabile di **salvare/caricare i propri dati**.
* Esempio: **Eloquent, CakePHP, Propel**

**🔸 Data Mapper**

* Separazione tra **logica dell’applicazione** e **persistenza dei dati**.
* Le entità non “sanno” nulla del DB.
* Esempio: **Doctrine**

**🔹 ORM più utilizzati:**

| **ORM** | **Pattern** | **Framework** |
| --- | --- | --- |
| Eloquent | Active Record | Laravel |
| Doctrine | Data Mapper | Symfony |
| CakePHP ORM | Active Record | CakePHP |
| Propel ORM | Active Record | Indipendente |

**🔹 Eloquent (Laravel)**

**⚙️ Caratteristiche principali:**

* **Active Record**: le classi rappresentano le tabelle del DB.
* **Convenzioni**:
  + Nome tabella in **plurale** (users)
  + Nome modello in **singolare** (User)
  + Tabelle pivot per relazioni N-N (role\_user)
* **Query Builder** integrato
* **Relazioni supportate**: 1-1, 1-N, N-N
* **Validazione dei dati**, **migrazioni**, **eventi Eloquent**

**🔹 Installazione Eloquent stand-alone (senza Laravel)**

php

CopiaModifica

composer require illuminate/database

**File di configurazione**:

php

CopiaModifica

use Illuminate\Database\Capsule\Manager as Capsule;

$capsule = new Capsule;

$capsule->addConnection([

'driver' => 'mysql',

'host' => 'localhost',

'database' => 'prova',

'username' => 'root',

'password' => 'Password&1',

]);

$capsule->bootEloquent();

**🔹 Relazioni in Eloquent**

**🔸 One-to-One**

php

CopiaModifica

class User extends Model {

public function phone() {

return $this->hasOne(Phone::class);

}

}

class Phone extends Model {

public function user() {

return $this->belongsTo(User::class);

}

}

**🔸 One-to-Many**

php

CopiaModifica

class Post extends Model {

public function comments() {

return $this->hasMany(Comment::class);

}

}

class Comment extends Model {

public function post() {

return $this->belongsTo(Post::class);

}

}

**🔸 Many-to-Many (Pivot Table)**

php

CopiaModifica

class User extends Model {

public function roles() {

return $this->belongsToMany(Role::class);

}

}

class Role extends Model {

public function users() {

return $this->belongsToMany(User::class);

}

}

**🔸 Accesso ai campi della tabella pivot:**

php

CopiaModifica

$utente->ruoli()->attach($ruoloId, ['valido' => true, 'data\_validita' => '2025-01-01']);

**🔹 Eloquent – DML (Data Manipulation Language)**

**🔸 Inserimento**

php

CopiaModifica

User::create(['username' => 'mario', 'password' => 'pwd']);

$user = new User;

$user->username = 'mario';

$user->password = 'pwd';

$user->save();

**🔸 Aggiornamento**

php

CopiaModifica

$user = User::find(1);

$user->username = 'Luigi';

$user->save();

**🔸 Cancellazione**

php

CopiaModifica

User::destroy(1); // oppure

$user = User::find(1);

$user->delete();

**🔸 Truncate**

php

CopiaModifica

User::truncate(); // svuota la tabella e azzera ID

**🔹 Query avanzate in Eloquent**

php

CopiaModifica

Flight::all(); // tutti i record

Flight::find(3); // id = 3

Flight::where('active', 1)->orderBy('name')->take(10)->get();

Flight::where('number', 'FR 900')->first();

Flight::where('active', 1)->count();

Flight::where('active', 1)->max('price');

User::find($id)->toArray();

**🔹 Eloquent – Creazione Modelli Personalizzati**

php

CopiaModifica

class User extends Model {

// Accessor

public function getNameAttribute($value) {

return ucfirst($value);

}

// Mutator

public function setPasswordAttribute($value) {

$this->attributes['password'] = bcrypt($value);

}

// Scope personalizzato

public function scopeActive($query) {

return $query->where('active', true);

}

}

**🔹 Eloquent – Esempio di Filtro Avanzato**

php

CopiaModifica

$books = Book::where('genre', 'Fantasy')

->where('cover\_type', 'hardcover')

->where('publish\_date', '>', '1999-01-01')

->orWhere('title', 'LIKE', '%rosa%')

->get();

**✅ Conclusione**

Eloquent semplifica **la gestione dei dati** in Laravel grazie a una sintassi pulita, alla gestione automatica delle relazioni e alla separazione tra logica e accesso ai dati. È ideale per sviluppatori che vogliono ridurre l’uso diretto di SQL pur mantenendo pieno controllo sul DB.

**🔹 COS’È AJAX**

**AJAX (Asynchronous JavaScript And XML)** è una tecnica per inviare e ricevere dati dal server in modo asincrono, senza ricaricare l'intera pagina.  
Non è un linguaggio di programmazione, ma **un insieme di tecnologie**:

* **JavaScript** per la logica.
* **XMLHttpRequest** (o fetch) per la comunicazione client-server.
* **DOM** per manipolare la pagina in base alla risposta del server.
* **JSON** (più moderno di XML) per lo scambio di dati.

**🔹 FUNZIONAMENTO DI BASE**

1. Il browser crea un oggetto XMLHttpRequest o usa fetch().
2. Viene aperta una connessione con open() (GET o POST).
3. Si inviano i dati con send().
4. JavaScript ascolta gli eventi come onreadystatechange.
5. Quando la risposta è pronta (readyState == 4), i dati vengono elaborati e visualizzati.

**🔹 XMLHttpRequest: METODI E STRUTTURA**

**1. Creazione**

js

CopiaModifica

var xhttp = new XMLHttpRequest();

**2. Apertura della richiesta**

js

CopiaModifica

xhttp.open("GET", "url", true); // true = asincrona

xhttp.open("POST", "url", true);

**3. Invio dei dati**

js

CopiaModifica

xhttp.send(); // per GET

xhttp.send("x=valore"); // per POST

**4. Impostare header personalizzati**

js

CopiaModifica

xhttp.setRequestHeader("Content-Type", "application/json");

xhttp.setRequestHeader("Authorization", "Bearer token123");

**🔹 STATO DELLA RICHIESTA: readyState e status**

* readyState:
  + 0 → Non inizializzata
  + 1 → Connessione aperta
  + 2 → Richiesta ricevuta
  + 3 → Elabora dati
  + 4 → Richiesta completata e risposta pronta
* status (HTTP):
  + 200 → OK
  + 403 → Forbidden
  + 404 → Not Found
  + 500 → Internal Server Error
  + …

**🔹 RISPOSTA DEL SERVER**

* responseText: risposta come stringa
* responseXML: risposta in formato XML
* getResponseHeader("Content-Type"): restituisce un header specifico
* getAllResponseHeaders(): restituisce tutti gli header

**🔹 ESEMPIO AJAX ASINCRONO**

js

CopiaModifica

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("GET", "ajax\_info.txt", true);

xhttp.send();

**🔹 FORMATI DEI DATI: JSON**

**JSON (JavaScript Object Notation)** è il formato più usato per scambiare dati:

json

CopiaModifica

{

"employees": [

{"firstName": "John", "lastName": "Doe"},

{"firstName": "Anna", "lastName": "Smith"}

]

}

**Parsing in JS:**

js

CopiaModifica

var obj = JSON.parse(jsonString);

**🔹 AJAX + JSON + DATABASE (PHP)**

**HTML/JS**

js

CopiaModifica

obj = { "name":"Mark", "limit":2 };

dbParam = JSON.stringify(obj);

xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xmlhttp.open("GET", "getData.php?x=" + dbParam, true);

xmlhttp.send();

**PHP (GET)**

php

CopiaModifica

header("Content-Type: application/json");

$obj = json\_decode($\_GET["x"]);

$conn = new mysqli("localhost", "root", "pass", "dbname");

$stmt = $conn->prepare("SELECT age FROM customer WHERE name=? LIMIT ?");

$stmt->bind\_param("ss", $obj->name, $obj->limit);

$stmt->execute();

$result = $stmt->get\_result();

echo json\_encode($result->fetch\_all(MYSQLI\_ASSOC));

**🔹 fetch() API – MODERNA ALTERNATIVA AD XMLHttpRequest**

**JS**

js

CopiaModifica

fetch("getData.php", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ username: "example" })

})

.then(response => response.json())

.then(data => {

console.log(data.username);

})

.catch(error => {

console.error("Error:", error);

});

**PHP**

php

CopiaModifica

$json = file\_get\_contents('php://input');

$data = json\_decode($json, true);

$response = ['message' => 'Dati OK', 'username' => $data['username']];

header("Content-Type: application/json");

echo json\_encode($response);

**🔹 CONCLUSIONI – VANTAGGI DI AJAX**

✅ Aggiornamento parziale della pagina  
✅ Esperienza utente migliorata  
✅ Riduzione del carico del server  
✅ Comunicazione fluida tra front-end e back-end  
✅ Perfettamente integrabile con PHP, JSON e database