Here's a complete PHP **Object-Oriented Programming (OOP)** example for managing records in a MySQL table with fields: `id`, `name`, `city`, and `phone`.

---

## □ Assumptions:

- Your MySQL table is named `contacts`.
- Table schema:

```
CREATE TABLE contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    city VARCHAR(100),
    phone VARCHAR(20)
);
```

---

## □ Database Class (DB connection)db.php file:-

```
<?php
class Database {
    private $host = "localhost";
    private $db_name = "your_database_name";
    private $username = "your_username";
    private $password = "your_password";
    public $conn;

    public function connect() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                "mysql:host={$this->host};dbname={$this->db_name}",
                $this->username,
                $this->password
            );
            $this->conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $e) {
            echo "Connection Error: " . $e->getMessage();
        }
        return $this->conn;
    }
}
?>
```

---

## □ Contact Class (CRUD operations) contact.php:-

```
<?php
class Contact {
    private $conn;
    private $table = "contacts";

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

    public function insert($name, $city, $phone) {
        $sql = "INSERT INTO {$this->table} (name, city, phone) VALUES (?, ?,
?)";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone]);
    }

    public function update($id, $name, $city, $phone) {
        $sql = "UPDATE {$this->table} SET name = ?, city = ?, phone = ? WHERE
id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone, $id]);
    }

    public function delete($id) {
        $sql = "DELETE FROM {$this->table} WHERE id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$id]);
    }

    public function getAll() {
        $sql = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($sql);
        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }

    public function search($keyword) {
        $sql = "SELECT * FROM {$this->table} WHERE name LIKE ? OR city LIKE ?
OR phone LIKE ?";
        $stmt = $this->conn->prepare($sql);
        $keyword = "%$keyword%";
        $stmt->execute([$keyword, $keyword, $keyword]);
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
}
?>
```

Here's a complete HTML + PHP solution to perform **CRUD (Create, Read, Update, Delete, and Search)** operations using forms.

---

## Folder Structure:

```
/crud_app/
│
├── db.php
├── contact.php
├── index.php          <-- Main UI (form + table)
├── process.php        <-- Handles insert/update/delete
```

---

## □ db.php – Database Connection

```
<?php
class Database {
    private $host = "localhost";
    private $db_name = "your_database_name";
    private $username = "your_username";
    private $password = "your_password";
    public $conn;

    public function connect() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                "mysql:host={$this->host};dbname={$this->db_name}",
                $this->username,
                $this->password
            );
            $this->conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $e) {
            echo "Connection Error: " . $e->getMessage();
        }
        return $this->conn;
    }
}
?>
```

---

## 📄 `contact.php` – Contact Class (CRUD)

```
<?php
class Contact {
    private $conn;
    private $table = "contacts";

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

    public function insert($name, $city, $phone) {
        $sql = "INSERT INTO {$this->table} (name, city, phone) VALUES (?, ?,
?)";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone]);
    }

    public function update($id, $name, $city, $phone) {
        $sql = "UPDATE {$this->table} SET name = ?, city = ?, phone = ? WHERE
id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone, $id]);
    }

    public function delete($id) {
        $sql = "DELETE FROM {$this->table} WHERE id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$id]);
    }

    public function getAll() {
        $sql = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($sql);
        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }

    public function getById($id) {
        $sql = "SELECT * FROM {$this->table} WHERE id = ?";
        $stmt = $this->conn->prepare($sql);
        $stmt->execute([$id]);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    public function search($keyword) {
        $sql = "SELECT * FROM {$this->table} WHERE name LIKE ? OR city LIKE ?
OR phone LIKE ?";
        $stmt = $this->conn->prepare($sql);
        $keyword = "%$keyword%";
        $stmt->execute([$keyword, $keyword, $keyword]);
```

```
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
}
?>
```

---

## □ `index.php` – HTML Form + Records Display

```
<?php
require_once 'db.php';
require_once 'contact.php';

$db = new Database();
$conn = $db->connect();
$contact = new Contact($conn);

$editData = null;
if (isset($_GET['edit'])) {
    $editData = $contact->getById($_GET['edit']);
}

$search = $_GET['search'] ?? '';
$records = $search ? $contact->search($search) : $contact->getAll();
?>

<!DOCTYPE html>
<html>
<head>
    <title>PHP OOP CRUD</title>
    <style>
        input, button { margin: 5px; padding: 5px; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        td, th { border: 1px solid #ccc; padding: 8px; }
    </style>
</head>
<body>
    <h2><?= $editData ? "Edit Contact" : "Add Contact" ?></h2>
    <form method="post" action="process.php">
        <input type="hidden" name="id" value="<?= $editData['id'] ?? '' ?>">
        <input type="text" name="name" placeholder="Name" value="<?=
$editData['name'] ?? '' ?>" required>
        <input type="text" name="city" placeholder="City" value="<?=
$editData['city'] ?? '' ?>" required>
        <input type="text" name="phone" placeholder="Phone" value="<?=
$editData['phone'] ?? '' ?>" required>
        <button type="submit" name="<?= $editData ? "update" : "add" ?>">
            <?= $editData ? "Update" : "Add" ?>
        </button>
```

```
    </form>

    <form method="get" action="">
        <input type="text" name="search" placeholder="Search..." value="<?=
$search ?>">
        <button type="submit">Search</button>
        <a href="index.php">Reset</a>
    </form>

    <h3>Contact List</h3>
    <table>
        <tr>

<th>ID</th><th>Name</th><th>City</th><th>Phone</th><th>Actions</th>
        </tr>
        <?php foreach ($records as $rec): ?>
        <tr>
            <td><?= $rec['id'] ?></td>
            <td><?= $rec['name'] ?></td>
            <td><?= $rec['city'] ?></td>
            <td><?= $rec['phone'] ?></td>
            <td>
                <a href="index.php?edit=<?= $rec['id'] ?>">Edit</a> |
                <a href="process.php?delete=<?= $rec['id'] ?>"
onclick="return confirm('Delete this contact?')">Delete</a>
            </td>
        </tr>
        <?php endforeach; ?>
    </table>
</body>
</html>
```

---

## □ `process.php` – Handle Form Submissions

```
<?php
require_once 'db.php';
require_once 'contact.php';

$db = new Database();
$conn = $db->connect();
$contact = new Contact($conn);

// Insert
if (isset($_POST['add'])) {
    $contact->insert($_POST['name'], $_POST['city'], $_POST['phone']);
}

// Update
if (isset($_POST['update'])) {
    $contact->update($_POST['id'], $_POST['name'], $_POST['city'],
$_POST['phone']);
}

// Delete
```

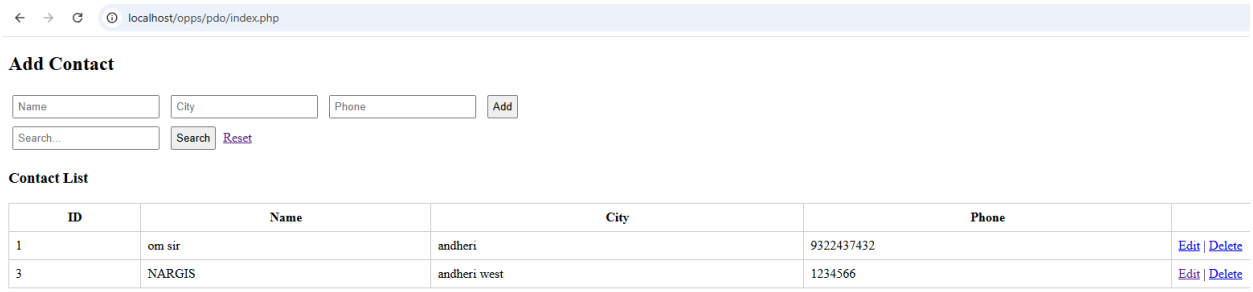
```php
if (isset($_GET['delete'])) {
    $contact->delete($_GET['delete']);
}

header("Location: index.php");
exit;
?>
```

---

## □ Final Steps:

1. Replace DB credentials in `db.php`.
2. Create the `contacts` table as shown earlier.
3. Place all files in a folder on your PHP server (e.g., XAMPP `htdocs`).
4. Access `http://localhost/crud_app/index.php` in your browser.



Here's a **line-by-line explanation** of the `db.php` file using **PDO** for database connection:

---

## `db.php` – Database Connection Class Using PDO

```
<?php
```

→ Starts the PHP script.

---

```
class Database {
```

→ Declares a class named `Database`. This class is used to manage the database connection in an object-oriented way.

---

```
    private $host = "localhost";
    private $db_name = "your_database_name";
    private $username = "your_username";
    private $password = "your_password";
```

→ These are **private properties** that hold the database configuration details:

- `$host` – the address of the database server (usually `localhost` when working locally).
- `$db_name` – the name of your MySQL database.
- `$username` – MySQL user (e.g. `root`).
- `$password` – MySQL user's password (often empty for `root` in XAMPP/WAMP).

These are `private` to ensure they cannot be accessed directly from outside the class.

---

```
    public $conn;
```

→ This is a **public property** that will hold the actual PDO connection object. It can be accessed from outside the class when needed.

---

```
    public function connect() {
```

→ Declares a **public method** called `connect()` that will be used to establish and return the PDO connection.

---

```
        $this->conn = null;
```

→ Initializes the `$conn` property to `null`. This ensures there is a default value before attempting the connection.

---

```
        try {
```

→ Starts a `try` block. If an error occurs while connecting, the script will jump to the `catch` block.

---

```
            $this->conn = new PDO(
                "mysql:host={$this->host};dbname={$this->db_name}",
                $this->username,
                $this->password
            );
```

→ This line attempts to create a new **PDO connection** to the MySQL database using:

- `mysql:host=...;dbname=...` → tells PDO to connect to a MySQL database with the given host and database name.
- `$this->username` and `$this->password` → credentials passed to authenticate the user.

If successful, `$this->conn` will hold the PDO connection object.

---

```
            $this->conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
```

→ Sets the PDO error mode to **Exception mode**. This means if there is an SQL error, it will throw a `PDOException` instead of silently failing. This is good for debugging and security.

---

```
        } catch(PDOException $e) {
```

→ If the connection fails, this block catches the exception (`PDOException`).

---

```
            echo "Connection Error: " . $e->getMessage();
```

→ Outputs a user-friendly message along with the actual error returned by PDO.

 In production, you'd typically **log** this error instead of showing it to the user.

```
        }
        return $this->conn;
    }
}
```

→ After the `try-catch` block:

- Returns the `$conn` object to be used elsewhere (e.g., by the `Contact` class).
- Closes the `connect()` method and the `Database` class.

---

```
?>
```

→ Ends the PHP script.

---

## Summary

| Line | Purpose |
|---|---|
| `class Database { ... }` | Encapsulates database connection logic. |
| `$host`, `$db_name`, `$username`, `$password` | Store connection credentials. |
| `$conn` | Will hold the PDO connection instance. |
| `connect()` | Main method that attempts to connect using PDO. |
| `try/catch` | Ensures graceful error handling on failure. |
| `setAttribute()` | Enables exceptions on query errors. |
| `return $this->conn` | Makes the connection accessible to other parts of your |

| Line | Purpose |
| --- | --- |
| | app. |

## `contact.php` – Contact Class for CRUD Operations

```
<?php
```

→ Opens the PHP script.

---

```
class Contact {
```

→ Defines a class named `Contact`. This class will handle operations (Create, Read, Update, Delete, and Search) on the `contacts` table.

---

```
    private $conn;
```

→ Declares a **private** property `$conn` to hold the database connection passed from outside (usually from `db.php`).

---

```
    private $table = "contacts";
```

→ Stores the name of the table in the database. This makes it easy to reuse or change table name later if needed.

---

```
    public function __construct($db) {
        $this->conn = $db;
    }
```

→ Constructor method:

- Called automatically when a new `Contact` object is created.
- It accepts the database connection `$db` and assigns it to `$this->conn`.

---

## ✅ `insert()` – Create a new contact

```
    public function insert($name, $city, $phone) {
        $sql = "INSERT INTO {$this->table} (name, city, phone) VALUES (?, ?,
?)";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone]);
    }
```

**Explanation**:

- SQL statement uses **parameterized query** (`?`) to prevent SQL injection.
- `$this->conn->prepare($sql)` prepares the SQL query.
- `execute([$name, $city, $phone])` binds the values to the placeholders and runs the query.

---

## ✅ `update()` – Update an existing contact

```
    public function update($id, $name, $city, $phone) {
        $sql = "UPDATE {$this->table} SET name = ?, city = ?, phone = ? WHERE
id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$name, $city, $phone, $id]);
    }
```

**Explanation**:

- Updates a contact's details by ID.
- Uses a parameterized query to safely pass values.
- Order of values in the `execute()` array matches the placeholders.

---

## ✅ `delete()` – Delete a contact

```
    public function delete($id) {
        $sql = "DELETE FROM {$this->table} WHERE id = ?";
        $stmt = $this->conn->prepare($sql);
        return $stmt->execute([$id]);
    }
```

**Explanation**:

- Deletes a record with a specific `id`.
- Uses a prepared statement for safety.

---

## 🔹 `getAll()` – Retrieve all contacts

```
    public function getAll() {
        $sql = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($sql);
        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
```

**Explanation**:

- Retrieves all rows from the `contacts` table.
- `fetchAll(PDO::FETCH_ASSOC)` returns each row as an associative array (`['id' => 1, 'name' => 'John', ...]`).

---

## 🔹 `getById()` – Get one contact by ID

```
    public function getById($id) {
        $sql = "SELECT * FROM {$this->table} WHERE id = ?";
        $stmt = $this->conn->prepare($sql);
        $stmt->execute([$id]);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }
```

**Explanation**:

- Retrieves a single contact by its `id`.
- `fetch(PDO::FETCH_ASSOC)` gets one row as an associative array.

---

## 🔹 `search()` – Search by name, city, or phone

```
    public function search($keyword) {
        $sql = "SELECT * FROM {$this->table} WHERE name LIKE ? OR city LIKE ? 
OR phone LIKE ?";
        $stmt = $this->conn->prepare($sql);
        $keyword = "%$keyword%";
        $stmt->execute([$keyword, $keyword, $keyword]);
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
```

**Explanation**:

- Performs a search in `name`, `city`, and `phone` columns.
- Adds `%` wildcards around the keyword for partial match.
- Uses `LIKE` with parameter binding to prevent SQL injection.

```
}
?>
```

→ Ends the `Contact` class and the PHP script.

---

## 🧾 Summary of Methods

| Method | Purpose |
|---|---|
| `insert()` | Add a new contact to the table |
| `update()` | Modify an existing contact by `id` |
| `delete()` | Remove a contact by `id` |
| `getAll()` | Fetch all contacts |
| `getById()` | Fetch one contact for editing |
| `search()` | Search contacts by keyword |

let's now go through a **line-by-line explanation** of the `index.php` file, which acts as the **main UI page** (form, search, and contact listing).

---

## 🧾 `index.php` – HTML Form + Contact List + Search

```
<?php
require_once 'db.php';
require_once 'contact.php';
```

- Includes the `db.php` and `contact.php` files to load the `Database` and `Contact` classes.
- This makes the database connection and contact logic available for use.

---

```
$db = new Database();
$conn = $db->connect();
$contact = new Contact($conn);
```

- Instantiates the `Database` class.
- Calls the `connect()` method to create a PDO connection.
- Passes the `$conn` object to the `Contact` class, which will use it for CRUD operations.

---

```
$editData = null;
if (isset($_GET['edit'])) {
    $editData = $contact->getById($_GET['edit']);
}
```

- Checks if an `edit` parameter exists in the URL (e.g., `index.php?edit=2`).
- If so, fetches that contact's data by ID to prefill the form for editing.
- Stores it in `$editData`.

---

```
$search = $_GET['search'] ?? '';
$records = $search ? $contact->search($search) : $contact->getAll();
```

- Grabs the `search` input from the URL query string.
- If a keyword exists, it performs a search; otherwise, it fetches all contacts.

---

## □ HTML Page Starts

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP OOP CRUD</title>
    <style>
        input, button { margin: 5px; padding: 5px; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        td, th { border: 1px solid #ccc; padding: 8px; }
    </style>
</head>
<body>
```

- Basic HTML structure and inline CSS styling for inputs, table, and buttons.

---

## □ Contact Form

```
<h2><?= $editData ? "Edit Contact" : "Add Contact" ?></h2>
<form method="post" action="process.php">
    <input type="hidden" name="id" value="<?= $editData['id'] ?? '' ?>">
    <input type="text" name="name" placeholder="Name" value="<?=
$editData['name'] ?? '' ?>" required>
    <input type="text" name="city" placeholder="City" value="<?=
$editData['city'] ?? '' ?>" required>
    <input type="text" name="phone" placeholder="Phone" value="<?=
$editData['phone'] ?? '' ?>" required>
    <button type="submit" name="<?= $editData ? "update" : "add" ?>">
        <?= $editData ? "Update" : "Add" ?>
    </button>
</form>
```

- **Form for adding or editing** a contact.
- If in edit mode:
  - The contact's data is pre-filled in the input fields.
  - A hidden `id` field is added for tracking the record being updated.
- Button name dynamically changes to `add` or `update` depending on the mode.

---

## □ Search Form

```
<form method="get" action="">
    <input type="text" name="search" placeholder="Search..." value="<?=
$search ?>">
    <button type="submit">Search</button>
    <a href="index.php">Reset</a>
</form>
```

- A basic search form using GET.
- If a search keyword was previously entered, it remains in the input.
- The **Reset** link clears the search and reloads the full list.

---

## □ Contact List Table

```
<h3>Contact List</h3>
<table>
    <tr>
        <th>ID</th><th>Name</th><th>City</th><th>Phone</th><th>Actions</th>
    </tr>
```

- Table headers.

```
php
CopyEdit
<?php foreach ($records as $rec): ?>
<tr>
    <td><?= $rec['id'] ?></td>
    <td><?= $rec['name'] ?></td>
    <td><?= $rec['city'] ?></td>
    <td><?= $rec['phone'] ?></td>
    <td>
        <a href="index.php?edit=<?= $rec['id'] ?>">Edit</a> |
        <a href="process.php?delete=<?= $rec['id'] ?>" onclick="return
confirm('Delete this contact?')">Delete</a>
    </td>
</tr>
<?php endforeach; ?>
```

- Loops through all fetched contact records and displays each one in a table row.
- Adds `Edit` and `Delete` links:
    - `Edit` reloads the same page in edit mode.
    - `Delete` points to `process.php` and asks for confirmation.

---

```
<?php if (empty($records)): ?>
<tr><td colspan="5">No contacts found.</td></tr>
<?php endif; ?>
</table>
```

- If no contacts are found, show a single-row message saying “No contacts found.”

---

```
</body>
</html>
```

- Closes the HTML body and document.

---

## ✅ Summary of Key Logic

| Section | Purpose |
|---|---|
| `require_once` | Loads classes and config. |
| `new Database()` | Connects to database via PDO. |
| `new Contact($conn)` | Sets up CRUD operations. |
| Form | Handles both add and edit operations. |
| Search | Filters the list of contacts. |
| Table | Displays all or filtered contacts. |
| Actions | Includes Edit and Delete functionality. |

Great! Let's walk through the **line-by-line explanation** of the 📄 `process.php` file, which handles:

- Form submission (Add / Update)
- Deletion of a contact
- Redirecting back to `index.php`

---

## 📄 `process.php` – Handles Add, Update, Delete

```
<?php
require_once 'db.php';
require_once 'contact.php';
```

- Includes `db.php` to set up the database connection.
- Includes `contact.php` to use the `Contact` class and its methods.

---

```
$db = new Database();
$conn = $db->connect();
```

```
$contact = new Contact($conn);
```

- Creates a `Database` object.
- Calls the `connect()` method to get the DB connection.
- Passes the connection to a new `Contact` object so we can perform CRUD operations.

---

## □ INSERT (Add New Contact)

```
if (isset($_POST['add'])) {
    $contact->insert($_POST['name'], $_POST['city'], $_POST['phone']);
}
```

- If the form was submitted with a button named `add` (i.e. we're adding a new contact),
- It calls the `insert()` method from `Contact` class.
- Passes the values from the form: `name`, `city`, and `phone`.

---

## □ UPDATE (Edit Existing Contact)

```
if (isset($_POST['update'])) {
    $contact->update($_POST['id'], $_POST['name'], $_POST['city'],
$_POST['phone']);
}
```

- If the form was submitted with a button named `update` (i.e. we're updating an existing contact),
- It calls `update()` with all four fields: `id`, `name`, `city`, and `phone`.

---

## □ DELETE (Delete Contact)

```
if (isset($_GET['delete'])) {
    $contact->delete($_GET['delete']);
}
```

- If a `delete` parameter is present in the URL (e.g., `process.php?delete=5`),
- The `delete()` method is called with that contact's ID.

---

## □ Redirect to `index.php`

```
header("Location: index.php");
```

```
exit;
```

- After insert, update, or delete — we **redirect** the user back to the main page (`index.php`).
- This avoids re-submitting the form if the user refreshes the page.
- `exit;` stops further script execution immediately.

---

## ✅ Summary

| Section | Trigger Type | Action |
|---|---|---|
| `isset($_POST['add'])` | Form submission | Calls `insert()` |
| `isset($_POST['update'])` | Form submission | Calls `update()` |
| `isset($_GET['delete'])` | Link click | Calls `delete()` |
| `header("Location: ...")` | Always | Redirects to main UI |

---

🎉 That completes the full explanation of all files in your CRUD app:

- `db.php` – DB connection
- `contact.php` – Class with all logic
- `index.php` – UI for form and listing
- `process.php` – Handles all operations