

# Fluent ORM Fundamentals

## 1. Basic Setup & CRUD

This demonstrates very basic CRUD operation.

```
// Initialize the Model
$userModel = new model('users');

// INSERT
$userModel->name = "Rakesh Shrestha";
$userModel->email = "rakesh@example.com";
$userModel->status = "active";
$id = $userModel->save();

// UPDATE (Fetch first, then change)
$user = $userModel->where('id', $id)->find();
if ($user) {
    $userModel->status = "verified";
    $userModel->save();
}
```

## 2. Data Filtering

This demonstrates the `whereGroup`, `whereIn`, and `whereBetween` logic.

```
$results = $userModel->select('id, name, email')
    ->where('status', 'active')
    ->whereIn('role_id', [1, 2, 3])
    ->whereBetween('created_at', '2025-01-01', '2025-12-31')
    ->whereGroup(function($q) {
        $q->where('name', 'LIKE', '%Dev%')
        ->orWhere('email', 'LIKE', '%admin%');
    })
    ->orderBy('name', 'ASC')
    ->limit(10)
    ->find();

foreach ($results as $row) {
    echo $row->name . " (" . $row->email . ")<br>";
}
```

### 3. Reporting with Aggregates

This is perfect for dashboard stats or internal reports.

```
$orderModel = new model('orders');
```

```
$report = $orderModel->selectRaw("customer_id, COUNT(id) as total_orders, SUM(total_amount) as revenue")
->join('customers', 'orders.customer_id', '=', 'customers.id')
->where('orders.status', 'completed')
->groupBy('customer_id')
->having('revenue', '>', 5000)
->orderBy('revenue', 'DESC')
->find();
```

### 4. The GraphQL-Style Query (Advanced)

This is perfect for a "User Profile" page where you want the user details and all their associated records (like posts or logs) in one go.

```
$userModel = new model('users');
```

```
// Define the schema: Keys are your JSON keys, Values are the table columns
```

```
$schema = [
  'id'      => 'id',
  'name'    => 'name',
  'email'   => 'email',
  'user_posts' => [
    'table'   => 'posts',
    'foreign_key' => 'user_id', // column in 'posts' table
    'fields'  => [
      'id'    => 'id',
      'title' => 'title',
      'body'  => 'content'
    ]
  ]
];
```

```
// Fetch user #1 with all their posts nested inside
```

```
$userWithPosts = $userModel->where('id', 1)->findGraph($schema);
```

```
// Accessing data:
```

```
echo $userWithPosts->name;
foreach ($userWithPosts->user_posts as $post) {
  echo $post->title;
}
```

## 5. Bulk Actions

No need to loop through records to delete or update them one by one.

```
// Bulk Update
$userModel->where('status', 1)
    ->where('created_at', '<', '2024-01-01')
    ->updateWhere(['status' => 'expired', 'archived' => 1]);

// Bulk Delete
$userModel->where('is_test_account', 1) -> deleteWhere();
```

## 6. Standard Pagination (Intermediate)

This is the most common way to display a list of records (like a user list or product catalog) with page links.

```
$userModel = new model('users');

// 1. Setup variables (usually from URL)
$currentPage = (int) ($_GET['page'] ?? 1);
$perPage = 10;

// 2. Build the query using our new extensions
$pagination = $userModel->select('p.id, p.name, p.email, roles.name as role_name')
    ->join('roles', 'p.role_id', '=', 'roles.id')
    ->where('p.status', 'active')
    ->search(['p.name', 'p.email'], $_GET['search'] ?? '') // Using the search extension
    ->orderBy('p.created_at', 'DESC')
    ->paginate($currentPage, $perPage);

/**
 * The $pagination object now contains:
 * $pagination->items -> Array of objects (the data)
 * $pagination->meta -> Object (total_records, total_pages, current_page, per_page)
 */

// 3. Display the Data
foreach ($pagination->items as $user) {
    echo "<div>{$user->name} - {$user->role_name}</div>";
}

// 4. Simple Pagination Links
echo links($pagination->meta)
```

## 7. Graph Pagination Sample (Advanced)

Use this when you need a complex JSON response for a frontend framework (like React or Vue) that includes related data.

```
$postModel = new model('posts');

// Define the nested relationship schema
$schema = [
    'post_id' => 'id',
    'post_title' => 'title',
    'author_id' => 'user_id',
    // Nesting Comments for each Post
    'comments' => [
        'table' => 'comments',
        'foreign_key' => 'post_id',
        'fields' => [
            'comment_id' => 'id',
            'comment_text' => 'body',
            'posted_at' => 'created_at'
        ]
    ]
];

// Fetch Paginated Posts with their Comments nested inside
$graphData = $postModel->where('status', 'published')
    ->orderBy('created_at', 'DESC')
    ->paginateGraph($schema, 1, 5);

// The resulting object is ready for json_encode()
header('Content-Type: application/json');
echo json_encode($graphData);
```

## 8. Table Joins and Transaction (Intermediate)

**Transactions** ensure that a group of database operations either all succeed or all fail together, while **Joins** allow you to pull data from related tables efficiently.

### 1. Transaction Feature

Transactions are essential for data integrity, such as when creating an Order and updating Product stock simultaneously. If the stock update fails, the Order is automatically rolled back.

```
$userModel = new model('users');
$logModel = new model('activity_logs');
```

```

try {
    $userModel->transaction(function() use ($userModel, $logModel) {
        // Operation 1: Create a user
        $userModel->name = "Jane Doe";
        $userModel->email = "jane@example.com";
        $userId = $userModel->save();

        // Operation 2: Create a log entry
        $logModel->user_id = $userId;
        $logModel->action = "Account Created";
        $logModel->save();

        // If anything throws an Exception inside this function,
        // the transaction rolls back automatically.
    });
    echo "Transaction successful!";
} catch (Exception $e) {
    echo "Transaction failed: " . $e->getMessage();
}

```

## 2. Join Feature

The `join()` method allows you to link tables. By default, it uses an `INNER JOIN`, but you can specify `'LEFT'`, `'RIGHT'`, or `'CROSS'`.

```
$postModel = new model('posts');
```

```

// Example: Get posts with their author's name from the 'users' table
$posts = $postModel->select('p.title, p.created_at, u.name as author_name')
    ->join('users u', 'p.user_id', '=', 'u.id')
    ->where('p.status', 'published')
    ->orderBy('p.created_at', 'DESC')
    ->find();

```

```

foreach ($posts as $post) {
    echo "Post: {$post->title} | Author: {$post->author_name}<br>";
}

```

## 3. Combining Join with Pagination

This is a very common real-world scenario—showing a paginated list of items with data from a related table.

```
$orderModel = new model('orders');
```

```

$results = $orderModel->select('p.*, c.name as customer_name, c.email as customer_email')
    ->join('customers c', 'p.customer_id', '=', 'c.id')
    ->where('p.total_amount', '>', 100)

```

```
->orderBy('p.id', 'DESC')
->paginate(1, 20);

// Use the data
foreach ($results->items as $order) {
    echo "Order #{ $order->id } for { $order->customer_name } ( { $order->total_amount } )<br>";
}
```