The process for secure database interactions using PDO(PHP Data Objects)

1. Establish a database connection: Start by creating a connection to the database using PDO. This involves specifying the database type, host, database name, username, and password. PDO will handle the connection and abstract the underlying database system.

Example->

```
$pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $username,
$password);
```

2. Prepare your SQL statements: Prepared statements separate the SQL logic from the data values, preventing potential SQL injection attacks. In a prepared statement, you use placeholders (e.g., `:name`, `:email`) in the SQL query to represent the data values.

Example->

```
$statement = $pdo->prepare("INSERT INTO users (name, email) VALUES (:name,
:email)");
```

3. Bind The Values::Bind the actual values to the placeholders using the `bindValue()` or `bindParam()` method, which ensures proper handling of the data types.

Example->

```
$statement->bindValue(':name', $name);
$statement->bindValue(':email', $email);
```

4. Execute the statement: Once the statement is prepared and the values are bound, execute the statement using the `execute()` method of the prepared statement object. PDO will send the SQL query and the bound values to the database for execution. This separation of SQL and data helps prevent SQL injection attacks.

Example->

```
$statement->execute();
```

5. Fetch the results (if applicable): If your query is expected to return results, use appropriate methods like `fetch()`, `fetchAll()`, or `fetchColumn()` to retrieve the data from the executed statement. These methods return the data in a structured format, such as an array or an object.

Example->

```
	$user = $statement->fetch(PDO::FETCH_ASSOC);
```

Additionally, PDO supports prepared statements, which provide an added layer of security by separating the SQL logic from the data values and preventing malicious input from altering the query's structure.