In the same folder with following PHP code we have SQLlite databse. File of the database is shopproduct.db. In the SQL we have table products. Column Names are id, type, name, surname, title, prioce, numpages, playlength

ChatGPT

Got it. If you have an SQLite database file named shopproduct.db in the same folder as your PHP code, and the database has a table named products with the specified column names, you can modify the fetchProductData function to fetch data from this SQLite database. Here's an updated version:

php

class ShopProductWriter {

// ... (Previous code)

function fetchProductData($productId) {

try {

// Create a PDO database connection

$db = new PDO('sqlite:shopproduct.db');

// Define the SQL query to fetch product data based on the product ID

$query = "SELECT \* FROM products WHERE id = :id";

// Prepare the SQL statement

$statement = $db->prepare($query);

// Bind the product ID parameter

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

// Execute the query

$statement->execute();

// Fetch the result as an associative array

$productData = $statement->fetch(PDO::FETCH\_ASSOC);

// Close the database connection

$db = null;

return $productData;

} catch (PDOException $e) {

// Handle any database connection errors

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

return false; // Or handle the error in a way that makes sense for your application

}

}

}

This code assumes that the SQLite database file (shopproduct.db) is in the same directory as your PHP script. Adjust the connection string (new PDO('sqlite:shopproduct.db')) if the database file is in a different location. Also, note that I added error handling using a try-catch block to catch any potential PDO exceptions. This can help you identify and handle errors more gracefully.



In the same folder with following PHP code we have SQLlite databse. File of the database is shopproduct.db. In the SQL we have table products. Column Names are id, type, name, surname, title, prioce, numpages, playlength and discount

ChatGPT

Great, with the updated information about your SQLite database structure, including the new columns type and discount, you can adjust the code accordingly. Here's an updated version of the fetchProductData function to include these new columns:

php

class ShopProductWriter {

// ... (Previous code)

function fetchProductData($productId) {

try {

// Create a PDO database connection

$db = new PDO('sqlite:shopproduct.db');

// Define the SQL query to fetch product data based on the product ID

$query = "SELECT \* FROM products WHERE id = :id";

// Prepare the SQL statement

$statement = $db->prepare($query);

// Bind the product ID parameter

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

// Execute the query

$statement->execute();

// Fetch the result as an associative array

$productData = $statement->fetch(PDO::FETCH\_ASSOC);

// Close the database connection

$db = null;

return $productData;

} catch (PDOException $e) {

// Handle any database connection errors

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

return false; // Or handle the error in a way that makes sense for your application

}

}

}

Now, the $productData array returned by this function should include the columns id, type, name, surname, title, price, numpages, playlength, and discount from your products table in the SQLite database.

You can then use these values as needed in your application. For example, you might update the ShopProduct class to include a method to set these values based on the fetched data.