Dynamic Web Application -MySQL Model

Book Cataloging System

Section 322, Group 6

# Table:

# 

# Insert Data and Test

# A screen shot of a computer Description automatically generatedA close-up of a computer code Description automatically generated

# Diagram

**daA screenshot of a computer

Description automatically generated**

# Dao files and Database PHP

# Dao files List：

1.BookDAO

2.CrudDAOInterface

3.UserbookDAO

4.UserDAO

5.BookCommentDao

# DatabasePHP：

6.Database.php

7.SQLHelper.php

1. BookDAO Description:

The BookDAO class is a concrete implementation of the CrudDAOInterface for managing book-related data interactions with the books table in the database. It encapsulates CRUD operations for books and uses the SqlHelper class to execute SQL queries.

**- Constructor (\_\_construct):**

Initializes a SqlHelper instance with the given $conn (database connection object) for executing SQL commands.

**- Insert (insert):**

Inserts the data from the passed $book object into the books table.

Catches and handles any exceptions, returning an array with the result of the operation.

**- Update (update):**

Updates a record in the books table corresponding to the id of the passed $book object.Catches and handles any exceptions, returning an array with the result of the operation.

**- Delete (delete):**

Deletes a record from the books table where the id matches the passed parameter.

Catches and handles any exceptions, returning an array with the result of the operation.

**- Find By ID (findById):**

Retrieves and returns a record from the books table where the id matches the passed parameter. Catches and handles any exceptions, returning an array with the result of the operation.

- **Find By Some Conditions (findBySomeConditions):**

Searches for book records in the books table that match certain criteria, such as title, author, ISBN, or genre, based on the input search keyword $text.

Returns an array containing book records or status information.

**- Find All (findAll):**

Retrieves all book records from the books table, ordered by creation time in descending order.Returns an array containing all book records or status information.

Exception handling in the code returns informative errors upon catching exceptions, aiding in debugging and error logging, thus preventing the application from failing due to unhandled exceptions.

1. CrudDAO Description:

Each method in the CrudDAOInterface corresponds to a basic operation one might need to perform at other DAO on a database record:

**insert($object):** This method is responsible for taking an object that contains data for a new record and saving it to the database. Implementations of this method will need to extract the data from the object and construct an SQL INSERT statement to create a new record.

**update($object): This** method should take an object that represents an existing record, including an identifier (like an ID), and update the corresponding record in the database. The SQL UPDATE statement will be used here.

**delete($id):** For deleting a record, this method uses the record's identifier to remove it from the database, often with an SQL DELETE statement.

**findById($id):** When you need to find a record by its unique identifier, this method will be used. It should execute a SELECT statement and return a single record matching the ID.

**findAll():** To retrieve all records from the corresponding database table, this method will typically run a SELECT statement without a WHERE clause, returning an array of all records.

1. UserBookDAO Description:

The UserBookDAO class handles database interactions related to the userbooks table.

The userbooks table is an association table linking users and books tables to keep track of which users own which books.

Here are the explanations for each method:

**Constructor (\_\_construct)**

Initializes a SqlHelper object, which is responsible for executing the actual SQL statements.

The constructor receives a database connection and uses it to initialize an instance of the SqlHelper class.

**Insert (insert)**

Before inserting data into the userbooks table, it checks whether the combination of userid and bookid already exists.

If it already exists, an error code and message are returned, indicating the book already exists.

If it does not exist, the insert operation is performed.

Any exceptions are caught and an array containing error information is returned.

The insert method attempts to add the combination of user and book to the userbooks table. If the combination already exists, a message indicating the book exists is returned; if the insert is successful, a success message is returned; if the operation fails or an exception occurs, an error is returned.

**Find By ID (findById)**

(Not implemented) This method should find a record in the userbooks table by its id.

The Find By ID method (not yet implemented) will search for a record in the userbooks table based on the ID.

**Update (update)**

(Not implemented) This method should update the record matching the provided object.

The update method (not yet implemented) will update an existing record in the userbooks table.

**Delete (delete)**

(Not implemented) This method should delete the record with the matching id.

The delete method (not yet implemented) will remove a record from the userbooks table.

**Delete By User ID And Book ID (deleteByUserIdAndBookId)**

Deletes the record in the userbooks table that matches the given userid and bookid.

If successful, a success message is returned; if the deletion fails, a failure message is returned.

This method deletes a record in the userbooks table based on user ID and book ID. If successful, a success message is returned; otherwise, a failure message is returned.

**Find By User ID And Book ID (findByUserIdAndBookid)**

Searches for a record in the userbooks table that matches the given userid and bookid.

If found, an array containing the record is returned; if not found, a message indicating the record does not exist is returned.

This method searches for a specific record in the userbooks table based on user ID and book ID. If found, record information is returned; if not found, it is indicated that the record does not exist.

**Find By User ID (findByUserId)**

Searches for all book records in the userbooks and books tables that match the given userid.

Returns an array containing detailed information about all the books owned by the user; if no records are found, a message indicating the record does not exist is returned.

This method searches for all book information collected by the user based on user ID and retrieves detailed information about these books from the books table.

**Find All (findAll)**

(Not implemented) This method should return all records in the userbooks table.

The Find All method (not yet implemented) should return all records in the userbooks table.

This DAO class's primary function is to enable a front-end application to manage a user's book collection. For example, when a user wants to add a book to their shelf, the insert method can be used. If a user wants to remove a book from their shelf, the deleteByUserIdAndBookId method can be used. All these methods use the SqlHelper class to simplify and unify database operations.

1. UserDAO Description:

The **UserDAO** class implements the **CrudDAOInterface**, managing data within the users table in the database. It offers basic CRUD operations and specific functions for querying users.

**Constructor (\_\_construct)**

Receives a database connection object, $conn, and uses this connection to initialize an instance of the SqlHelper class. This instance will be used for executing all SQL operations later on.

Insert (insert)

Before inserting user information into the users table, it first checks if a user with the same name already exists.

If the user already exists, it returns an array containing an error code and message, indicating the user already exists.

If the user does not exist, it performs the insertion operation, adding the new user's username and password into the table.

If the insertion operation is successful, it returns an array containing a success message; if it fails, it returns an array with a failure message.

If an exception occurs during the operation, it catches the exception and returns an array containing the error information.

**Update (update)**

(Not implemented) This method should update the record matching the provided user object's id.

**Delete (delete)**

(Not implemented) This method should delete a record from the users table by its id.

**Find By ID (findById)**

(Not implemented) This method should find and return a record from the users table by its id.

**Find By Username (findByUserName)**

This method is used to find a user in the users table by username.

If a matching user is found, it returns an array containing the user's information; if not found, it returns a message indicating the record does not exist.

**Find All (findAll)**

(Not implemented) This method should return all user records from the users table.

This class's functionality allows the application to perform operations on user data, such as registering new users or finding specific users. By using SqlHelper to execute actual SQL queries, it simplifies interactions with the database

1. BookCommentDAO Description:

The BookCommentDAO class implements the CrudDAOInterface, designed to manage the data in the bookcomments table of the database. This table stores information on book reviews, including which user commented on which book. Here is an explanation of the methods in this class:

**Constructor (\_\_construct)**

Receives a database connection object $conn and uses it to initialize an instance of the SqlHelper class. The SqlHelper is used to perform actual SQL query operations.

Insert (insert)

This method is for inserting new book review information into the bookcomments table.

It accepts a $bookcomment object as a parameter, which contains the book ID (bookid), user ID (userid), and comment content (comments).

If the insert operation is successful, it will return an array with a success message; if the insert fails, it returns an array with a failure message.

If an exception occurs during the operation, it catches the exception and returns an array containing the error information.

**Update (update)**

(Not implemented) As required by the CrudDAOInterface, this method should update an existing comment record, but it has not been implemented in the current class definition.

**Delete (delete)**

(Not implemented) This method should delete a record in the bookcomments table by its unique identifier (ID), but it has not been implemented in the current class definition.

**Find By ID (findById)**

(Not implemented) This method should find and return a comment record in the bookcomments table by its ID, but it has not been implemented in the current class definition.

**Find By Book ID (findByBookId)**

This method is used to find and return all comment records associated with a specific book ID (bookid).

It performs an SQL query, joining the bookcomments table with the users table to retrieve the comments as well as the usernames of the users who made the comments.

If the query is successful and relevant records are found, it returns an array containing these records; if no records are found, it returns a message indicating that the record does not exist.

**Find All (findAll)**

(Not implemented) This method should return all comment records in the bookcomments table, but it has not been implemented in the current class definition.

The primary function of the BookCommentDAO class is to allow the application to perform CRUD operations on book comments, including adding new comments and querying all comments for a specific book. By utilizing the SqlHelper for executing SQL queries, it simplifies the interaction process with the database.

# DatabasePHP：

1. Database.php

The Database.php file defines a Database class designed for database connection and management. Here's a breakdown of the main components and their functions:

mysqli\_report()

This line of code sets MySQLi error reporting to throw mysqli\_sql\_exception exceptions in case of errors or strict SQL issues instead of just warnings or silent failures. This aids in better debugging and error handling.

Private Properties

The class defines several private properties to store the necessary information for database connection, such as server name ($servername), username ($username), password ($password), and database name ($dbname).

**Constructor (\_\_construct)**

The constructor is private, a key feature of the singleton pattern, preventing external code from directly instantiating the Database class.

Inside the constructor, it creates a new MySQLi object to connect to the database. If the connection fails, the script execution is terminated, and the connection error is reported.

**getInstance()**

This is a public static method to obtain the unique instance of the Database class. If the instance does not exist yet (i.e., $instance is null), it will create a new Database instance and store it in the $instance static property. Subsequent calls will return this already created instance, ensuring the uniqueness of the database connection.

**getConnection()**

This public method returns the database connection object ($conn). This allows other parts of the backend code to access and use this connection for executing SQL queries through the instance of the Database class.

In summary, the Database.php file, through the singleton pattern, ensures that there will only be one instance of the database connection throughout the entire application, thus avoiding unnecessary repeated connections. This class is a part of the backend components, aiding backend services in interacting with the MySQL database. The frontend interacts with the database indirectly through APIs provided by the backend.

1. SQLHelper.php

The SqlHelper class provides a convenient method for executing database queries and operations, while also handling the logic of prepared statements and parameter binding. It encapsulates the details of interacting with the database, making the execution of SQL statements more concise and secure.

**Constructor (\_\_construct)**

Accepts a database connection object $conn and stores it as a private property of the class. This connection object is used for all subsequent database operations.

**executeQuery($sql, $params = [], $isSelect = true)**

This method is used to execute SQL queries.

The $sql parameter is the SQL statement to be executed.

The $params is an array containing the values for the parameters in the SQL statement, used for binding in the prepared statement to prevent SQL injection attacks. It defaults to an empty array.

The $isSelect parameter specifies whether this query is a SELECT query (i.e., whether it expects a result set). It defaults to true.

If $isSelect is true, the method executes the query and returns an array containing all rows in the result set. Each row is an associative array with column names as keys.

If $isSelect is false, the method executes a non-query statement (such as INSERT, UPDATE, or DELETE) and returns the number of affected rows.

**createObjectFromRow($className, $row)**

This static method accepts a class name $className and an associative array $row containing a row from the database query result.

The method checks if the class named $className exists. If it does not, an exception is thrown.

Then, the method creates an instance of the $className class and sets the instance's properties using the data in $row. It accomplishes this by dynamically constructing setter method names and checking if those methods exist. If a corresponding setter method exists, it is called with the value from the row.

Finally, it returns the instance of the object filled with data.

This class is a backend tool for database operations, enhancing code security through prepared statements and parameter binding, and adding flexibility through dynamic object creation. Using SqlHelper simplifies the database operation code, making it easier to maintain and extend.