```php
<?php

// Create connection
$conn = mysqli_connect("localhost","root","","tanmay");
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) { //////
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

The provided code snippet attempts to insert multiple records into a table called "MyGuests" using the mysqli_multi_query function in PHP. Here's a breakdown and potential improvements:

Functionality:

1. **$sql variable:** It concatenates three individual INSERT statements for different users ("John Doe", "Mary Moe", and "Julie Dooley").
2. **mysqli_multi_query($conn, $sql)**: This line tries to execute all three INSERT statements sequentially in the provided connection ($conn).
3. **if statement**: It checks if mysqli_multi_query returned true, indicating success.
4. **echo "New records created successfully";**: Prints a success message if the execution was successful.
5. **else block**: If the execution failed, it prints "Error" and displays both the concatenated SQL statements ($sql) and the specific MySQL error message using mysqli_error($conn).

Potential improvements:

- **Error handling:** While mysqli_multi_query provides a single error message for all queries, it's difficult to pinpoint the specific failing entry. Consider using a loop to execute each INSERT statement individually and handle errors for each one separately.
- **Transaction management:** If all insertions need to succeed or fail together, consider wrapping them within a transaction using mysqli_begin_transaction and mysqli_commit or mysqli_rollback based on the outcome.
- **Prepared statements:** For improved security and performance, consider using prepared statements instead of concatenating the SQL directly. This helps prevent SQL injection vulnerabilities and improves query execution efficiency.
- **Logging:** Implement logging mechanisms to record successful and failed operations with details like timestamps, user information, and specific errors.