Key Points:

1. **Parameterized Query:**
    - A `PreparedStatement` uses placeholders (`?`) in the SQL query to represent values.
    - Example:

    ```
    SELECT * FROM Books WHERE Book_No = ? AND Book_Name = ?;
    ```

    - This allows you to pass user inputs (data) separately without embedding them directly into the query.
2. **SQL Injection Prevention:**
    - By separating query structure and input, `PreparedStatement` ensures that user-supplied values are treated strictly as data, not executable SQL code.
    - The database engine automatically escapes special characters (e.g., quotes), neutralizing injection attempts.
    - Example Injection Attempt: `Book_Name = 'a' OR '1'='1'`
        - With `PreparedStatement`, this input will be treated as a literal string, not as part of the SQL logic.
3. **Precompilation:**
    - The query is precompiled by the database engine when the `PreparedStatement` is created.
    - This means the database analyzes, parses, and optimizes the query once, and you can execute it multiple times with different values efficiently.
4. **Code Reusability:**
    - Once a `PreparedStatement` is created, it can be reused with different parameters without needing to rewrite or reconstruct the query.
    - Example:

    ```
    pstmt.setInt(1, 101);
    pstmt.setString(2, "Java Basics");
    pstmt.executeQuery();

    pstmt.setInt(1, 102);
    pstmt.setString(2, "Advanced Java");
    pstmt.executeQuery();
    ```

5. **Type-Safe Parameter Binding:**
    - You explicitly bind data types to the placeholders (`?`) using methods like `setInt()`, `setString()`, etc. This ensures that the data matches the expected type in the database schema.
6. **Improved Readability and Debugging:**
    - Instead of concatenating user inputs into a query string, you use a clean, readable SQL template.

- Debugging tools can show the complete query with parameter values substituted, making it easier to trace issues.
