MilWare – Military Database System 1.0

Author: David Čadek

Date: January 9, 2026

TEST SCENARIO No. 2: Verification of Functionality and Data Manipulation

1. Test Objective

The goal is to verify the system's main functions: creating and modifying data (CRUD), working with relationships between tables (Missions), and importing data from external files, including user modification of input data (JSON).

2. Prerequisites (What you need prepared)

- The application is successfully installed (see Scenario No. 1).
- The database milware_db exists and is populated with the basic structure (init_db.sql).
- The application is running in the command line (python src/main.py) and the Main Menu is visible.
- In the database, open the views_db file in the same way as init_db.sql, and execute it again using the yellow lightning bolt icon.

3. Test Procedure (Step-by-Step)

Part A: Import and Data Modification Test (JSON)

Data Preparation (File Modification):

1. Keep the application running and switch to Windows Explorer.
2. Open the project folder -> data folder.
3. Right-click on the new_recruits.json file and select **Open with -> Notepad** (or another editor).
4. Add or modify one soldier. For example, change the name "Soap" to "Test Subject" (or add a new block {...}).
5. Save the file (Ctrl+S) and close it.

Executing Import in the Application:

1. Return to the running application (black window).
2. In the main menu, press 8 (**IMPORT DATA**) and confirm with Enter.
3. The program asks what to import. Select 1 (**Import SOLDIERS**).

4. **Expected Result:** The program prints [SUCCESS] Loaded X soldiers.

Result Verification:

1. In the main menu, press 1 (**DATA LISTING**) -> 1 (**Soldiers**).
2. Look for your "Test Subject" in the table.
3. **Check:** If present, the JSON file import is working correctly.

Part B: CRUD Operations (Creating and Editing a Soldier)

We will verify that we can create, change, and delete data directly in the application.

Creation (CREATE):

1. In the menu, select 3 (**Recruit new soldier**).
2. **Name:** Terminator T-800
3. **Callsign:** Arnold
4. **Rank:** General
5. **Base ID:** 1
6. **Result:** The program confirms saving and assigns an ID.
7. *It is possible to verify by listing the list using command 1*.

Update (UPDATE):

1. In the menu, select 4 (**Promote/Edit**).
2. Enter the ID of the soldier you just created (Terminator).
3. For **"New Name"**, just press Enter (no change).
4. For **"New Rank"**, type: Private (we will perform a demotion).
5. **Result:** The program confirms the update.

Check:

1. In the menu, select 2 (**Find Soldier**). Enter their ID.
2. Verify that they have the rank Private.

Part C: M:N Relationship Test (Mission)

Preparation: Ensure you have at least one soldier and one mission in the database (if not, import or create them).

Assignment:

1. In the menu, select 6 (**SEND SOLDIER ON MISSION**).
2. Select the ID of any soldier.
3. Select the ID of any mission.
4. For the role, type: Cook.
5. **Result:** The program prints [SUCCESS] Order confirmed.

Verification in Database:

1. In the menu, select 1 (**DATA LISTING**).
2. Select option 5 (**Orders / MissionAssignments**).
3. You must see a row in the table with your soldier's name, the mission name, and the role "Cook".

Part D: Reports and Statistics

We will verify that the application can calculate aggregated data from multiple tables.

1. In the menu, select 7 (**GENERAL REPORT**).
2. Check the printed table.
3. The "Soldiers" column must not be zero (if you imported data).
4. The "Vehicles" column should correspond to the number of imported vehicles.

Part E: TRANSACTION TEST (Mission Assignment and Soldier Modification)

Preparation and Default State Check:

1. Select one soldier (e.g., "Soap") and remember their ID and Callsign (must be without an asterisk).
2. **State before action:** Soldier is not on the mission, Callsign is clean.

Executing Transaction:

1. In the menu, select 6 (**SEND SOLDIER ON MISSION**).
2. Enter the ID of the selected soldier ("Soap").
3. Select the ID of any mission.
4. Enter role: Commander.
5. **Expected Result:** The program prints [SUCCESS] Order confirmed. Transaction executed.

Verification of changes in TWO tables:

1. **Step A (Table 1 - Assignments):**
    - Go to menu 1 -> 5 (**Orders**).
    - Verify that a new row with your soldier and role "Commander" has been added to the list.
2. **Step B (Table 2 - Soldiers):**
    - Go to menu 1 -> 1 (**Soldiers**).
    - Find your soldier.
    - **Key Check:** Their Callsign must have changed – it must now contain an asterisk (e.g., Soap*).
3. **Conclusion:** The operation successfully modified data in two independent tables within a single transaction.

4. Test Procedure – Error and Exception Testing

In this section, we intentionally attempt to enter incorrect data. The goal is to prove that the application does not crash with a system error (Python Traceback) but instead displays a comprehensible error message.

Test 1: Invalid Input (Text instead of Number)

The most common user error.

1. In the main menu, select **2** (Find soldier by ID).
2. The program expects a number (ID). Instead, type text: abc and press Enter.
3. **Expected Reaction:**
    - The application **MUST NOT CRASH**.
    - It must print a message like: *"Error: Invalid value entered, please enter a number."*
    - It should allow retrying the choice or returning to the menu.

Test 2: Non-existent Menu Option

1. In the Main Menu, enter a number that does not exist, e.g., **99**.
2. **Expected Reaction:**
    - The program prints: *"Invalid choice"* and redraws the menu. The program does not terminate.

Test 3: Logical Error (Deleting non-existent ID)

1. In the menu, select **5** (Dismiss soldier).
2. Enter an ID that is not in the database (e.g., 58964).
3. **Expected Reaction:**
    - The program prints: *"Soldier with ID 58964 not found"* (or *"Record does not exist"*).
    - The program does not crash on a database error.

Test 4: Crisis Scenario – Database Failure

Tests the handling of try-except blocks during database communication.

1. Leave the application running in the menu.
2. Switch to Windows and **stop the MySQL service** (via XAMPP Stop or in Workbench "Stop Server").
3. In the application, try to list soldiers (Menu **1** -> **1**).
4. **Expected Reaction:**
    - The program **MUST NOT CRASH** with a red code dump.
    - It must catch the mysql.connector.Error exception and print: *"Database connection error"* or *"Connection refused"*.
    - The program safely terminates or returns to the menu.

5. Test Evaluation

The test is considered PASSED if:

- [ ] Data was successfully loaded from the JSON file you manually modified.
- [ ] The newly created soldier (Terminator) was found in the system and could be edited.
- [ ] Mission assignment proceeded without error and the record appeared in the relationship table listing.
- [ ] The application did not crash (terminate with an error log) once during the test.

The test FAILED if:

- Import did not load changes made in Notepad (incorrect JSON format).
- Application crashed when entering ID (e.g., typing a letter instead of a number).