MilWare – Military Database System 1.0

Author: David Čadek

Date: January 9, 2026

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

Obsah

1. Test Objective

The goal of this scenario is to verify system stability during administrative tasks. The test simulates the work of a database administrator performing bulk imports, manual data corrections (CRUD), assigning complex tasks (Missions), and verifying if the system reacts correctly to non-standard inputs and infrastructure failures.

2. Prerequisites (What you need prepared)

- The application is installed and running (python src/main.py).
- The database milware_db is active and contains the basic structure (init_db.sql).
- Views are active in the database (views_db).
- You have access to edit files in the data folder.

3. Test Procedure (Step-by-Step)

Part A: External Data Validation (JSON Import)

We will verify if the system accepts changes in data files made "from outside".

1. **Data Preparation:**
    - Leave the application running in the background.
    - In the data folder, open the file new_vehicles.json (or soldiers.json depending on your setup).
    - **Action:** Find any record and change the full_name item to **"Admin Testovací"**.
    - Save the file and close it.
2. **Import:**
    - In the application, select menu **8** (IMPORT DATA).
    - Select **1** (Import SOLDIERS).
    - **Expected Result:** The program prints [SUCCESS].
3. **Verification:**
    - In the menu, select **1** (LISTING) -> **1** (Soldiers).
    - Check if the table contains the soldier **"Admin Testovací"**.
    - **Check:** Data loads dynamically, import works.

Part B: Personnel Management (CRUD Operations)

Simulation of recruiting a new specialist and their subsequent modification.

1. **Creation (CREATE):**
    - Menu **3** (Recruit).
    - **Name:** Chuck Norris
    - **Callsign:** Ranger
    - **Rank:** Private
    - **Base ID:** 1
    - **Result:** Program confirms ID assignment.
2. **Update (UPDATE):**
    - Menu **4** (Promote/Edit).
    - Enter the ID of the new soldier (Chuck Norris).
    - **New Name:** (Press Enter - no change).
    - **New Rank:** General (immediate promotion).
    - **Result:** Program confirms the update.
3. **Check:**
    - Menu **2** (Find soldier) -> enter ID. Verify that the rank is General.

Part C: Deployment to Action (M:N Relationship)

1. **Assignment:**
    - Menu **6** (SEND ON MISSION).
    - Select the ID of the soldier (Chuck Norris) and the ID of a mission.
    - **Role:** Chief Negotiator.
    - **Result:** Program prints [SUCCESS] Order confirmed.
2. **Relationship Check:**
    - Menu **1** -> **5** (Orders). The record must exist.

Part D: Overviews and SQL Views

1. Menu **7** (GENERAL REPORT).
2. Check if the soldier counts for Base No. 1 are summing correctly (Chuck Norris should be included).
3. Verify that the report does not output SQL query errors.

Part E: TRANSACTION TEST (Data Consistency)

Key Test: Changing state in two tables simultaneously.

1. **Initial State:**
    - Select a soldier with a "clean" name (e.g., "Admin Testovací" from the import). Verify there is no asterisk in their Callsign.
2. **Transaction:**
    - Menu **6** (SEND ON MISSION).
    - Select this soldier and any mission.
    - **Role:** Observer.
    - **Result:** [SUCCESS] Transaction executed.
3. **Cross-Check (Proof of Transaction):**
    - **Table 1 (Missions):** Menu **1**->**5**. A record with the role "Observer" exists.
    - **Table 2 (Soldiers):** Menu **1**->**1**. The soldier now has the Callsign **Admin Testovací*** (with an asterisk).
    - **Conclusion:** The system successfully performed an atomic operation across multiple tables.

4. Test Procedure – Error and Exception Testing

Stress test of application resilience.

Test 1: Data Type Handling

1. In the menu, select **5** (Dismiss soldier).
2. The program waits for a number (ID). Enter text: chyba (error).
3. **Reaction:** The application MUST NOT CRASH. It must print *"Invalid value, please enter a number"* and return to the menu.

Test 2: Non-existent Menu Choices

1. In the main menu, enter 007.
2. **Reaction:** The application prints *"Invalid choice"* and redraws the menu.

Test 3: Data Integrity (Non-existent Record)

1. Menu **4** (Edit soldier).
2. Enter ID 999999.
3. **Reaction:** The application prints *"Soldier not found"* (not a database error).

Test 4: Simulation of Connection Failure (Connection Error)

1. Leave the application running.
2. In Windows (XAMPP/Workbench), **stop the MySQL service**.
3. In the application, select **1** -> **1** (List soldiers).
4. **Reaction:**
    - The application **MUST NOT CRASH** (Python Traceback).
    - It must capture the exception and print: *"Database connection error"* (or similar message from the try-except block).
    - The application terminates correctly or returns to the menu.

5. Test Evaluation

The test is evaluated as PASSED if:

- [ ] Imported data (Admin Testovací) is visible in the system.
- [ ] CRUD operations (Creating Chuck Norris) were performed and data is stored in the DB.
- [ ] The transaction proceeded correctly (Soldier is on a mission AND has an asterisk in their name).
- [ ] The application withstood text input instead of a number (did not crash).
- [ ] The application withstood a database outage with a user-friendly error message.

The test is evaluated as FAILED if:

- The application crashed (Traceback) during any step.
- The transaction was only partially executed (e.g., asterisk is missing).
- Imported data did not appear.