MilWare – Technical Documentation

Project Name: MilWare – Military Database System 1.0

Version: 1.0 **Author:**

David Čadek **Date:** January 10, 2026

Obsah

1. Project Overview

MilWare is a console-based database application designed for the management of military personnel, vehicle fleets, and operational missions. The system allows authorized users to track soldiers and vehicles, assign them to specific bases, and manage their deployment in missions.

The application is built using **Python** (for the application logic) and **MySQL** (for data storage). It demonstrates key database concepts such as CRUD operations, M:N relationships, Transaction processing (ACID), and Data Aggregation.

1.1 Key Features

- **Personnel Management:** Recruit, promote, demote, and dismiss soldiers.
- **Asset Management:** Track military vehicles and their capacity.
- **Mission Control:** Assign soldiers to missions with specific roles (M:N relationship).
- **Data Import:** Bulk import of data from external JSON files.
- **Reporting:** Generate aggregated reports on base statistics.
- **System Integrity:** Transactional processing ensures data consistency; error handling prevents crashes.

2. System Architecture & Technologies

2.1 Technology Stack

- **Programming Language:** Python 3.8+
- **Database Engine:** MySQL Server (8.0+)
- **Database Connector:** mysql-connector-python
- **Configuration:** JSON (config.json)
- **Development Environment:** VS Code / MySQL Workbench

2.2 Project Structure

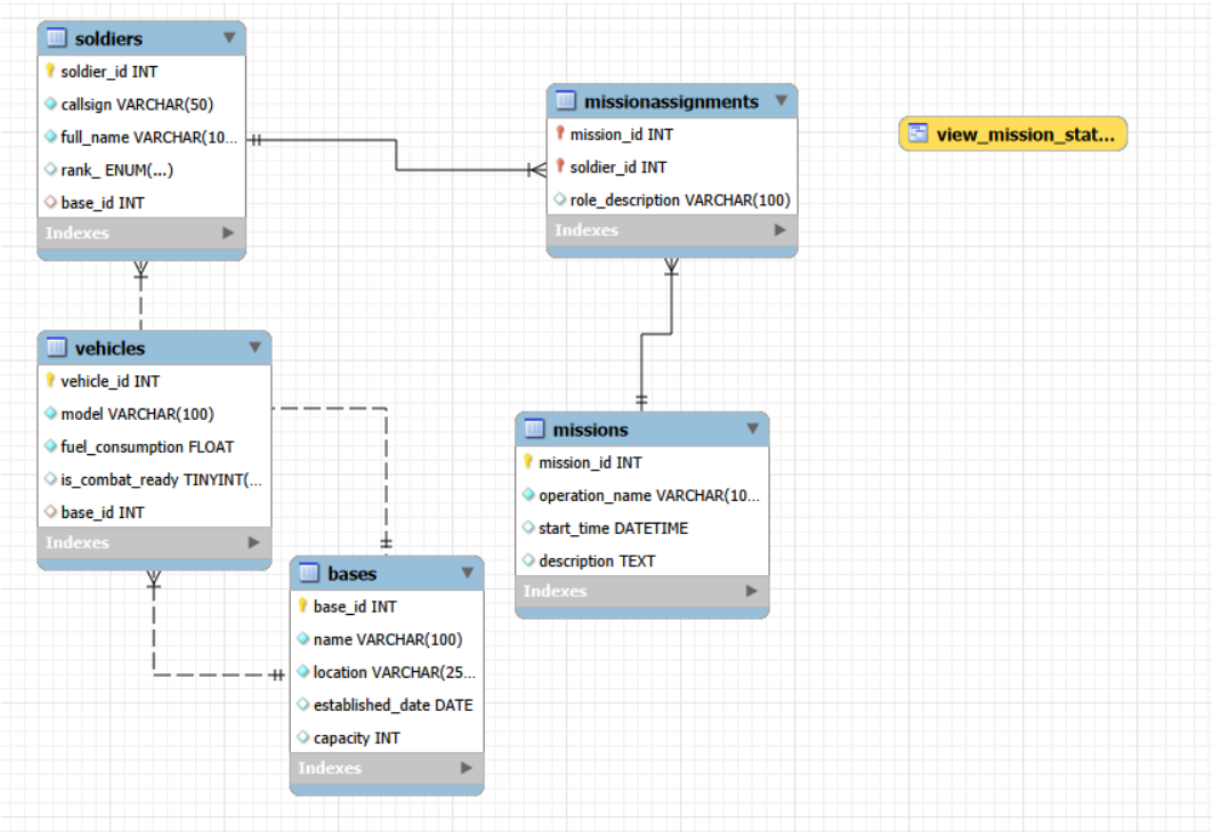
The project follows a modular architecture separating the database logic from the user interface:

- **src/** – Source code folder.
    - main.py – Entry point, handles the Main Menu and UI loop.
    - database.py – Handles MySQL connection using the **Singleton pattern**.
    - **repositories/** – Contains classes for database operations (DAO pattern):
        - soldier_repository.py – CRUD for Soldiers.
        - vehicle_repository.py – CRUD for Vehicles.
        - mission_repository.py – Handles Missions and Assignments.
        - view_repository.py – Handles SQL Views and Reports.
- **data/** – JSON files for data import (soldiers.json, vehicles.json).
- **sql/** – SQL scripts for database initialization (init_db.sql).
- **config.json** – Database connection settings (Host, User, Password).

3. Database Design

The database milware_db consists of normalized tables linked by foreign keys to ensure data integrity.

3.1 Entity-Relationship Diagram (ERD)



3.2 Database Schema

1. **Bases**: Stores military base locations.
    - *Columns:* id (PK), name, location, capacity.
2. **Soldiers**: Stores personnel data.
    - *Columns:* id (PK), full_name, callsign, rank, base_id (FK).
3. **Vehicles**: Stores vehicle data.
    - *Columns:* id (PK), type, model, base_id (FK).
4. **Missions**: Stores mission details.
    - *Columns:* id (PK), name, difficulty, status.
5. **MissionAssignments** (Association Table): Implements the **M:N relationship** between Soldiers and Missions.
    - *Columns:* id (PK), soldier_id (FK), mission_id (FK), role (e.g., Commander, Medic).

3.3 SQL Views

The system utilizes SQL Views to simplify complex queries:

- **View_Soldier_Details**: Joins Soldiers and Bases to display the base name instead of the ID.
- **View_Base_Stats**: Aggregates data to show the count of soldiers and vehicles per base.

4. Implementation Details

4.1 CRUD Operations

The application implements full CRUD (Create, Read, Update, Delete) for the main entities. SQL injection protection is ensured by using parameterized queries (e.g., cursor.execute(query, (value1, value2))).

4.2 Transaction Management (ACID)

Transactions are used to ensure data consistency, particularly in the **"Send on Mission"** feature.

- **Scenario:** When a soldier is assigned to a mission, the system:
    1. Inserts a record into MissionAssignments.
    2. Updates the Soldiers table to append an asterisk (*) to the soldier's callsign.
- **Implementation:** If either step fails, a ROLLBACK is triggered, reverting all changes. If successful, COMMIT saves the data.

4.3 Error Handling & Configuration

- **Configuration:** The database connection is not hardcoded. It is loaded from config.json. If the file is missing, the application alerts the user safely.
- **Exceptions:** The code wraps database operations in try-except blocks. It catches errors like mysql.connector.Error (connection failure) or invalid inputs, preventing the application from crashing and displaying a user-friendly message instead.

4.4 Data Import

The system supports bulk data import from JSON files. This feature uses transactions to ensure that if a JSON file contains invalid data (e.g., negative capacity), the entire import is rolled back to maintain database purity.

5. Installation Guide

To set up the MilWare system on a new machine:

1. **Database Setup:**
    - Install MySQL Server (or XAMPP).
    - Open MySQL Workbench and execute the script sql/init_db.sql.
2. **Python Environment:**
    - Install Python 3.8+.
    - Install dependencies: pip install mysql-connector-python.
3. **Configuration:**
    - Rename config.example.json to config.json.
    - Edit the file to match your MySQL credentials (user/password).
4. **Execution:**
    - Run the application via terminal: python src/main.py.

Conclusion**:** MilWare successfully meets all defined requirements for a robust database application. It demonstrates secure database connectivity, advanced SQL features, and resilient Python programming practices suitable for a production-ready prototype.