Volvo Rental Manager (D1)

Author: Nikola Poláchová

Contact: polachova@spsejecna.cz

Date: 30. 12. 2025

School: SPŠE Ječná

1. Project Specification

The goal of this application is to simulate the management of a construction machinery rental fleet, specifically focusing on Volvo equipment. The system allows for tracking machines, managing customer data, and recording complex rental transactions.

- **Functional Requirements:**
  - Inventory management (adding machines and categories).
  - Customer relationship management.
  - Transactional rental processing (M:N relationship).
  - Automated data import and aggregate reporting.
- **Technical Requirements:**
  - Python 3.11+.
  - Oracle Database (19c/21c XE).
  - Libraries: `oracledb`, `tkinter`, `shutil`, `json`, `os`, `sys`.

2. Architecture & Design Patterns

This project implements the **Data Access Object (DAO)** pattern, fulfilling the **D1** assignment requirement.

- **Presentation Layer:** The `MainMenu` class (built with `tkinter`) handles the GUI and user inputs.
- **Logic Layer:** Definition of objects like `Machine`, `Customer`, `RentalItem`, and `Rental`.
- **Data Access Layer:** Classes `MachineDAO`, `CustomerDAO`, and `RentalDAO` encapsulate all SQL communication.

3. Database Model (E-R Diagram)

The database consists of 5 tables connected via relational links, including a Many-to-Many (M:N) relationship between Rentals and Machines.

1. **Categories:** Stores machine types and power sources (Enum: diesel, electric, hybrid).
2. **Customers:** Stores client information (String, Date).

3. **Machines:** Stores equipment details (Model, Float weight, Boolean availability).
4. **Rentals:** Header table for rental agreements.
5. **Rental_Items:** Junction table for the M:N relationship (stores price and duration).

Sure, here is the complete documentation and test scenarios translated into English. I have styled it to look like a well-structured student project report.

4. Advanced Database Features

- **Views:** The system uses two views: `V_Available_Machines` (for quick fleet overview) and `V_Customer_Revenue_Report` (for aggregated financial data).
- **Transactions:** The "Confirm Rental" process is handled as a multi-table transaction. It inserts data into `Rentals` and `Rental_Items` while updating the `Machines` table. It uses `commit()` on success and `rollback()` on failure.
- **Import:** On startup, the `setup()` function automatically imports data from CSV files into three tables.

5. Error Handling & Validation

The application handles various failure states:

- **Configuration Errors:** Handles missing or incorrect `config.json` files.
- **Connection Errors:** Catches database connection failures and notifies the user.
- **Input Validation:** Prevents invalid data entry (e.g., non-numeric values in price fields) using GUI alerts.

6. Checklist

1. [Requirement 1: Real RDBMS] - The application uses a professional Oracle Database system. Connection is handled via the oracledb library in "Thin mode", which fulfills the requirement for a real relational database system.
2. [Requirement 2: DB Structure (5+ Tables, 2x View, 1x M:N)] - The database contains exactly 5 tables (Categories, Customers, Machines, Rentals, Rental_Items). It includes a Many-to-Many (M:N) relationship via the junction table Rental_Items. Two views, Available_Machines and Customer_Revenue_Report, are automatically created during the setup() process.
3. [Requirement 3: Data Types (Float, Bool, Enum, String, Date/Time)] - Every required data type is represented in the schema: Float (weight), Boolean

equivalent (is_available via check constraint), Enum (power via check constraint), String (VARCHAR2), and Date/Time (registration_date and created timestamp).

4. [Requirement 4: Multi-table CRUD] - The software allows the user to insert and display information that spans multiple tables. Specifically, creating a new rental records data into Rentals and Rental_Items while simultaneously updating the machine's availability in the Machines table.
5. [Requirement 5: Multi-table Transactions] - A transaction is programmed for the rental process. It involves writing to Rentals and Rental_Items and updating Machines. The code uses connection.commit() on success and connection.rollback() in the except block to ensure data integrity across these tables.
6. [Requirement 6: Aggregate Report] - The application generates a summary report using the V_Customer_Revenue_Report view. This report performs SUM and COUNT operations while joining three tables (Customers, Rentals, Rental_Items) to show meaningful business data.
7. [Requirement 7: CSV Data Import] - The import_data() function in the DBSetup.py module handles the import of data into three different tables (Categories, Customers, Machines) from CSV files located in the /Data directory.
8. [Requirement 8: Configuration File] - The program is configured via an external config.json file. Users can set their database user, password, and dsn without needing to modify the source code.
9. [Requirement 9: Error Handling & Validation] - All inputs are validated, and the code includes specific try-except blocks for configuration errors, database connection failures, and missing files. Errors are communicated to the user via descriptive messagebox alerts.

7. Test Scenarios

Test Case 1: Installation and Database Initialization

- **Setup:** Ensure Oracle XE is running and `config.json` is provided.
- **Action:** Launch the application with an incorrect DB password in `config.json`.
- **Expected Result:** A "Connection Error" message box appears with specific details.
- **Action:** Correct the password and launch again.
- **Expected Result:** The console prints "Tables created" and "Success data imported". The main UI loads with data populated from CSVs.

Test Case 2: Creating a Rental (M:N Transaction)

- **Action:** Open "New Rental", select a customer and an available machine. Enter days and price. Click "Confirm".
- **Expected Result:** Records are added to `Rentals` and `Rental_Items`. The machine's `is_available` status in the DB changes to 0.
- **UI Validation:** The machine list in the main window updates automatically; the rented machine either disappears from the available list or changes color.

Test Case 3: Input Validation and Aggregate Reporting

- **Action:** Attempt to create a new machine with a blank model name or a non-numeric weight.
- **Expected Result:** A warning dialog appears. No data is sent to the database.
- **Action:** Access the "Reports" section.
- **Expected Result:** The system displays aggregate data (e.g., total revenue per customer) fetched from the `V_Customer_Revenue_Report` view.

Instructions for the Tester:

1. **Repository:** Clone the project from GitHub.
2. **Configuration:** Rename `config.json.example` to `config.json` and enter your Oracle connection information.
3. **Data:** Do not move the `/Data` folder, as the automated setup depends on it.
4. **Execution:** Run `main.py` or use the pre-built `main.exe` in the `/dist` directory.