A Comprehensive Database Solution for MUC's Used Car Operations

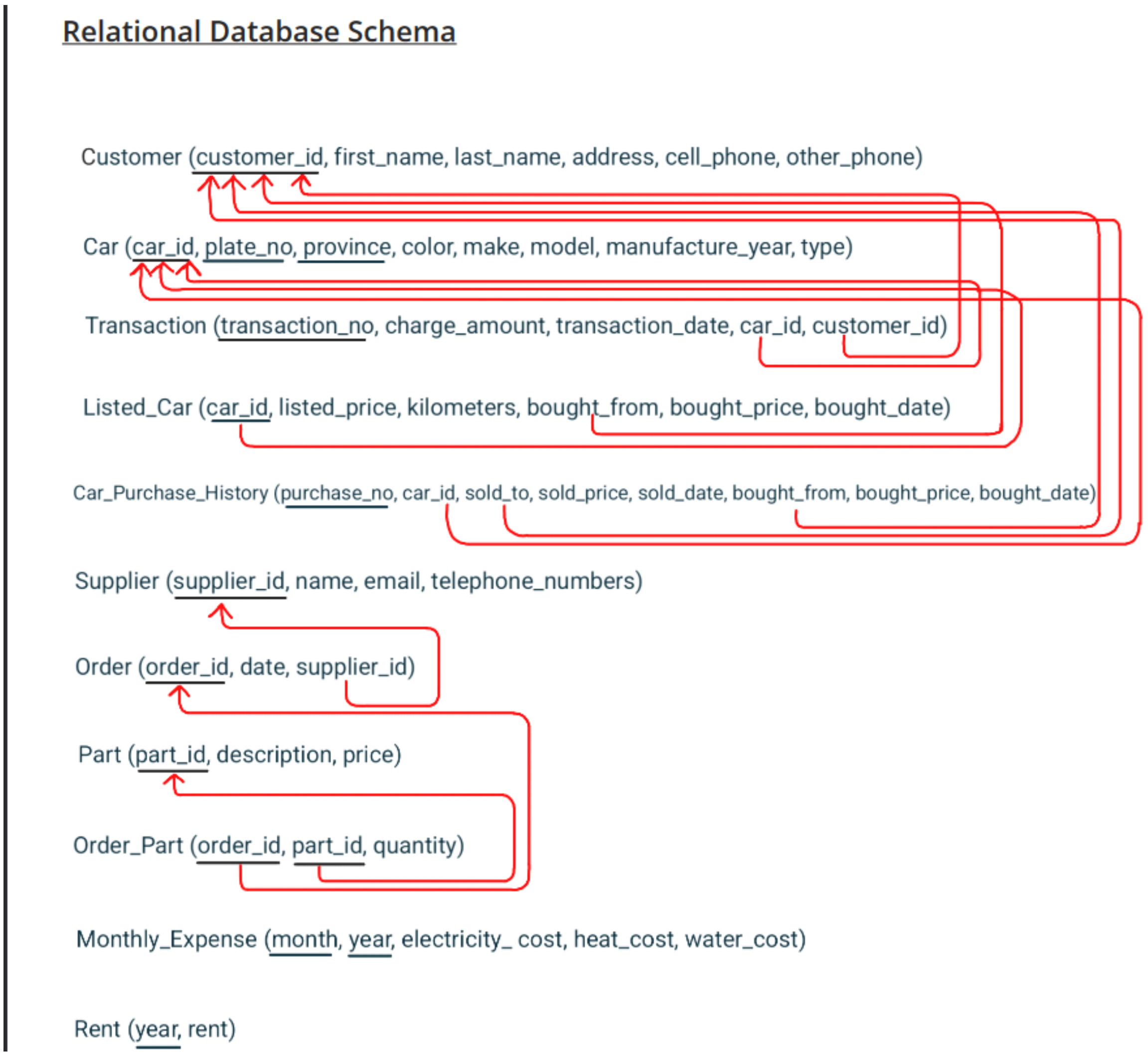
Members

Sheikh Saad Abdullah (A00447871)
Naziya Tasnim (A00447506)
Mohak Shrivastava (A00445470)
Kazi Istiak (A00452666)

Introduction

The MUC (Motors Used Cars) project involves designing and implementing a comprehensive database system for a national company specializing in used cars. The database aims to efficiently manage customer information, vehicle data, parts inventory, supplier details, transaction records, and monthly expenses. This project is a group effort, requiring collaboration among team members to design a robust relational database schema and develop a web application that interacts with the database to perform various operations.

Database Schema



Relational Database Schema

Customer (customer_id, first_name, last_name, address, cell_phone, other_phone)

Car (car_id, plate_no, province, color, make, model, manufacture_year, type)

Transaction (transaction_no, charge_amount, transaction_date, car_id, customer_id)

Listed_Car (car_id, listed_price, kilometers, bought_from, bought_price, bought_date)

Car_Purchase_History (purchase_no, car_id, sold_to, sold_price, sold_date, bought_from, bought_price, bought_date)

Supplier (supplier_id, name, email, telephone_numbers)

Order (order_id, date, supplier_id)

Part (part_id, description, price)

Order_Part (order_id, part_id, quantity)

Monthly_Expense (month, year, electricity_ cost, heat_cost, water_cost)

Rent (year, rent)

The MUC database schema, as depicted in the image, is a well-structured relational model that includes several tables representing different entities. These tables are: Customer, Car, Transaction, Listed_Car, Car_Purchase_History, Supplier, Order, Part, Order_Part, Monthly_Expense, and Rent. Each table has its own primary keys, which are unique identifiers for the records in the table. The relationships between these tables are established through foreign keys, which are fields in a table that match the primary key of another table. This schema provides a robust framework for managing and manipulating data in the MUC system, ensuring data integrity and consistency. It allows for efficient data retrieval and updates, making it a vital component of the system’s overall performance and reliability.

Web Application Structure:

1. **Main Files :**
    - HTML → Within this category we have 5 files
        - budget.html → this HTML code is a template for a web page that displays a budget for a certain number of years, with the budget adjusted for inflation.
        - error.html → his HTML code is a template for a basic error page that informs the user about an error that occurred during their request.
        - expenses.html → this HTML code is a template for a web page that displays expenses for a range of years, with the expenses for each year shown in a table.
        - index.html → This HTML code provides a basic user interface for interacting with a database, allowing users to perform operations such as showing a table, adding a supplier, summarizing annual expenses for parts, and projecting a budget based on the number of years and inflation rate.
        - table.html → this HTML code is a template for displaying a table with dynamic content, where you can specify the table name, column headers, and table data to be displayed.
    - CSS → The HTML code includes a link to an external CSS file with the URL `https://unpkg.com/mvp.css`. This means that the styles for the table and other elements on the page are defined in the `mvp.css` file, which is hosted on the unpkg CDN. [1]
    - Python → we have 3 main python files that we need.
        - __main__.py → This Python script is a command-line application that serves as an entry point for a web application. It uses argparse to parse command-line arguments and getpass to securely prompt for a MySQL password.
        - app.py → This Python code defines a factory function that returns a Flask web application with several routes for interacting with a MySQL database by adding suppliers, viewing expenses, projecting budgets and displaying tables

- db.py → This Python code defines a context manager class `Database` for handling MySQL database connections. It ensures that connections are properly established and closed, and transactions are committed or rolled back as needed, while also handling any exceptions that may occur.

- DButils → files for the database interaction
  - j2sql_parts.sh → The script defines variables `user`, `pass`, and `db` to store the MongoDB and MySQL database credentials. Imports data from a JSON file (`parts_100.json`) into a MongoDB database. Exports data from the MongoDB collection (`parts`) to a CSV file (`parts.csv`). `mysql`: Imports data from the CSV file (`parts.csv`) into a MySQL database (`$db`) and table (`parts`). The `cat parts.csv | tr "," "\t" > parts.tsv` command converts the CSV file (`parts.csv`) to a tab-separated values (TSV) file (`parts.tsv`). The `load data local infile 'parts.tsv' into table parts` command loads data from the TSV file (`parts.tsv`) into the MySQL `parts` table.
  - j2sql_supp+order.sh → This Bash script performs several operations related to creating tables in a MySQL database, converting JSON data to TSV files, and loading the data into the MySQL database.
  - j2tsv_supp+order.py → This Python script converts JSON data from two files (suppliers and orders) into tab-separated values (TSV) files.
  - make_tables.sql → This SQL code defines the schema for four tables: `suppliers`, `suppliers_telephone`, `orders`, and `order_parts`, along with their respective columns and relationships.
  - orders_4000.json → JSON document containing a list of objects, each representing an order.
  - parts_100.json → JSON document containing a list of objects, each representing an item.
  - parts_table.sql → SQL statement that creates a table named `parts` if it does not already exist.

2. **Database Interaction :**

   Web Interface (HTML, CSS, JS):

   - You have several HTML files (`index.html`, `error.html`, `table.html`, `expenses.html`, `budget.html`) that define the structure of your web interface.
   - CSS (`mvp.css`) is used for styling the HTML elements.

   Flask Application (`app.py`):

     - This file defines your Flask application and its routes.
     - Each route (`/`, `/table`, `/supplier`, `/expenses`, `/budget`) corresponds to a different functionality in your web application.
     - The functions defined for each route handle the request, interact with the database, and render the appropriate HTML templates.

   Database Interaction (`db.py` and `app.py`):

     - The `db.py` file defines a `Database` class that serves as a context manager for handling database connections.
     - In `app.py`, the functions for each route use this `Database` class to interact with the MySQL database.
     - SQL queries are executed to retrieve data (e.g., table information, expenses) and insert data (e.g., add a supplier).

   Database Setup (`make_tables.sql`):

   - This file contains SQL statements to create tables in the MySQL database (`suppliers`, `suppliers_telephone`, `orders`, `order_parts`, `parts`).

   Data Import (`import_data.sh`):

     - This shell script is used to import data into the MySQL database.
     - It uses `mongoimport` to import data from JSON files into MongoDB and then exports it to CSV files.
     - Finally, it loads the CSV files into the MySQL database.

   Data Conversion (`j2tsv_supp+order.py`):

   - This Python script converts JSON data into TSV format for `suppliers` and `orders`.

Conclusion

In conclusion, the project demonstrates the use of web development technologies (Flask, HTML, CSS), database management (MySQL, MongoDB), and scripting (Python, Bash) to create a functional web application for managing a database. It showcases how different components can work together to build a robust system for handling data and providing useful functionalities to users.

References

[1] https://unpkg.com/mvp.css@1.14.0/mvp.css (`/* MVP.css v1.14 - https://github.com/andybrewer/mvp */`)