Carter Williams

Team 7

11/20/22

Vehicle Management System

This project implements a system that is intended to help a car dealership store and interact with data related to their business. The data stored includes records of customers, employees, departments, vehicles, and vehicle sales. Obviously, this does not cover all the data storage and interaction requirements for a car dealership, but it focuses on a large portion of their data requirements. Dealership employees can perform CRUD operations on the data through a web interface implemented with Django. The system is not meant to be accessed by the dealership’s customer’s because it functions only as a management system for employees, inventory, and transactions.

This project meets the requirements because it implements seven tables and a total of ten relationships among these tables. This count only includes the tables I implemented, not the default Django database tables.

Conceptual Schema (EER Diagram)

Diagram

Description automatically generated

Fig 1. EER diagram for the system. Provides data visualization

with little to no regard for implementation details.

The EER diagram depicted above shows the data requirements for the system at a high abstraction level. Entities are depicted as text inside rectangles, like User or Vehicle, and relationships among the entities are depicted as lines connecting the two. Customer and Employee are sub entities of User because they share common attributes but have their own respective attributes. Truck and Car are sub entities of Vehicle for the same reason. Customer, Employee, and Vehicle all relate to a sale entity. This entity is a weak entity because it would not exist without the three entities’ relationships. A customer, employee, and vehicle are all involved in the sale of a vehicle, so they are required for the relationship. One customer can buy many vehicles, and one employee can sell many vehicles. On the other hand, one vehicle can only be sold once. Bubbles connected to each entity represent attributes that describe the entity.

Database Schema

Diagram

Description automatically generated

Fig 2. Schema for database that depicts tables, columns, and keys.

This is the current schema for the system that depicts how the relational database is laid out. I will not discuss tables that do not start with “vehiclemanagement” because they are boilerplate tables created by Django. The tables that begin with “vehiclemanagement” were designed and implemented by me.

In the schema depicted, columns have a symbol to the left of them that describe what the column is. A lightning bolt symbolizes a primary key, a red symbol represents a foreign key, and a diamond simply represents a column. The tables implemented in the database include vehicle, employee, car, truck, vehiclemodel, department, and vehiclesale. A more detailed description of the purpose, normal form, and characteristics of each table will be discussed in the next section.

Detailed Description

This section will go into further detail about the database and the data it holds. There will be three topics discussed: the data requirements, normal forms, and core functional requirements of the database.

Data Requirements

The database is meant to hold information relevant to a part of the dealership’s business. This includes their inventory of vehicles, employee/customer information, and the sale of vehicles. The requirements of data do not include things like auto-loan information, vehicle repair data, or exhaustive details about a vehicle. Simply put, the database is required to store employee, customer, vehicle, and vehicle sale data.

These requirements are met by implementing seven tables (not including the default Django tables). Firstly, the customer data is handled solely by a Django table, auth\_user. It stores basic information about a user, like email and name. Employee data is comprised of three tables: auth\_user, employee, and department. The employee table is a sub-entity of auth\_user that adds relevant information about an employee. This includes their social security number, birthday, phone number, salary, and other information. Department is a separate table responsible for holding a department’s name and manager.

Vehicle data is stored with four tables: vehicle, vehiclemodel car, and truck. Vehicle is a super-entity of car and truck that stores relevant information about all vehicles. This includes data like the sticker price, color, vin number, a picture, and other information. Vehiclemodel is a table that relates to vehicle and stores the make, model, year, and type of vehicle. Finally, car and truck are sub-entities of vehicle that store pertinent information respective to their type. For example, car stores the max speed and number of seats, but truck stores the area of the bed and towing capacity.

Finally, vehicle sale data is stored in the vehiclesale table. This table simply holds the price and date of the sale. It also stores the foreign keys for the vehicle sold, customer who bought the vehicle, and the employee who sold the vehicle.

The implemented database meets all the data requirements outlined in this section. For further detail about the implementation, such as all the columns and their types, view the schema diagram or models.py file in the source code. A description of the individual tables will also be given in the “Normal Forms” subsection.

Normal Forms

This section will cover the normal forms of the tables implemented by me. The default Django tables will not be covered because I cannot change them and did not create them.

1. Employee

The employee table contains a single-column primary key (which is also a foreign key to auth\_user) and two, single-column foreign keys. The primary key is implemented as an incrementing integer. There is only one other candidate key in this table, SSN. This table is in Boyce-Codd normal form because every column only depends on a unique column. Each column only depends on the primary key or SSN column because no column is functionally determined by another column in this table. For example, the birthday column only depends on the primary or candidate key column because the person’s birthday cannot functionally be determined by their sex, phone number, salary, commissions, manager, and so on.

1. Department

The department table contains a single-column primary key and one, single-column foreign key. The primary key is implemented as an incrementing integer. There is only one other candidate key in this table, the department name. This table is also in Boyce-Codd normal form because every column only depends on a unique column. The primary key only depends on itself or the name, the name depends only on the primary key, and the manager only depends on the name or primary key.

1. VehicleModel

VehicleModel contains a contains a single-column primary key, and the primary key is implemented as an incrementing integer. There is a composite candidate key which includes (make, model, and year). This table is only in second normal form because the only non-key column, vehicle type, depends on the model column. Vehicle type still depends on the whole primary key, but it also depends on a non-unique column. This table performs better in second normal form, and was left there intentionally, because this table is read from much more frequently than it is updated. By keeping this data stored in the same tablespace, it increases the SELECT query’s speed on this table.

1. Vehicle

Vehicle contains a contains a single-column primary key, and it is the vehicle’s vin number. There is also a foreign key that references the vehiclemodel table. There are no other candidate keys in this table. This table is in Boyce-Codd normal form because all columns depend only on a unique column, the primary key. The only potential argument that could be made that would knock this table down to second normal form is if the (vehiclemodel, trim) determine the sticker price. In this system, we are assuming that the sticker price is determined by the dealership, so it is not the same for all vehicles with the same make, model, year, and trim level. This is due to other nuances like extra packages, factory damage, or other aspects of the vehicle.

1. Car

The car table contains a single-column primary key which is also a foreign key to the vehicle table. There are no other candidate keys, so the primary key is the only unique column. This table is in Boyce-Codd normal form because the other two columns, max speed and number of seats, each depend only on the primary key. Neither of them depends on the other; therefore, all columns depend only on a unique column.

1. Truck

The truck table contains a single-column primary key which is also a foreign key to the vehicle table. There are no other candidate keys, so the primary key is the only unique column. This table is in Boyce-Codd normal form because the other two columns, bed area and towing capacity, each depend only on the primary key. Neither of them depends on the other; therefore, all columns depend only on a unique column.

1. VehicleSale

The vehiclesale table contains a single-column primary key, and this key is an auto incrementing integer. There are three single-column foreign keys that each refer back to the employee that sold the vehicle, the customer that bough it, and the vehicle sold. The composite of these three columns is also a candidate key. This table is in Boyce-Codd normal form because each column depends only on the unique column, the primary key. No column in this table, other than the primary key, can functionally determine another one.

Core Functional Requirements

The requirements of the database were discussed in the previous two subsections, so this section will focus on the requirements of the system. In other words, how the user can view, change, and add data to the database. Firstly, this system is not a complete system for a car dealership, meaning not all the functionality is there. This is mainly due to the time constraint and requirements of this project.

The first functional requirement of the system is to allow the user to login and interact with all the other functionalities, and this functionality is only available to employees. Closely related to logging in, a user must be able to register a new account, whether that be a customer or employee. This function is only available to employees, so current employees must register new employees or customers. Employees can also edit and view customers and employees.

The second set of functional requirements includes interacting with data related to vehicles. This includes creating, updating, and viewing vehicles. The search function of vehicles allows employees to filter by if the vehicle is sold and the type of vehicle (car or truck). Employees can also add, update, or view a sale of a vehicle.

The final set of functional requirements includes viewing miscellaneous information about the data in the database. This includes total sale amount, employee salary, and total inventory size displayed on the home screen. These numbers are displayed for the employee to provide analytical insight into the financials and inventory of the dealership.

Development Environment

The development environment of this system can be divided into three components: the database, backend, and frontend. The database is the component that is responsible for storing the data on the persistent storage device for later use. The backend is responsible for handling HTTP requests made by the user to interact with the database. Finally, the frontend is responsible for gathering user input and sending HTTP requests to the backend.

First, the database development environment included MySQL, and it handles all the database queries. The database uses the default storage engine, InnoDB. Specific details about the physical design of the database will be discusses in the next section.

Next, the backend development environment included many tools and packages that helped simplify communication with the database. The main package used included Django, a package written in Python that includes an ORM and HTTP web framework [0]. Django simplifies database queries by implementing table data as Python objects that come with methods to perform queries. For example, the Python statement “Employee.objects.all()” equates to “SELECT \* FROM Employee;” in SQL. This is just one example, but Django provides many method for executing joins, updates, inserts, deletions, and many more. Using an ORM allows for interaction with a SQL database in the native language for the application, Python in my case. Django does allow for raw queries to be written, and this comes in handy when writing uncommon queries. For example, summing the total amount of customer purchases in a certain year. Django’s HTTP web framework provides URL configuration and HTTP request parsing. Another package used for the backend includes MySQLclient. This is the driver Django uses in order to send queries to the database. The final package used for the backend is Pillow, a package that simplifies the process of saving files to system storage. This package was used exclusively for saving images of the vehicles.

The frontend development environment includes tools and packages that help with writing HTML, CSS, and JavaScript. Django also helps in this regard with “Django templates,” and this is a tool for inserting information into HTML files before they are sent to the client. This system works by passing variables to the HTML file from the function that renders the file, and Django parses the file prior to it being sent to the client. While parsing the HTML file, Django changes these variables to their values passed from the function. This tool also provides for loops, if statements, and many other unique tools for rendering HTML pages dynamically. A tool used for providing styling includes Bootstrap. This is a package that comes with many CSS classes and styles, and it allows for quicker web site design by providing basic CSS. Finally, Chart.js was used for a revenue chart on the website’s home page.

Evaluation of the System

No changes to the default implementation of table structure was made to the MySQL database. This means these things were implemented by InnoDB on every table: [1]

* All indexes are B+ tree.
* Each primary key has a primary index.
* Each foreign key has a secondary index.
* Tables have a sorted structure on the primary key.
* Each table is contained within its own tablespace.

Only one additional index was added manually, and this was a secondary index on the “sale\_date” column located in the vehiclesale table. I added this index because many of the analytical queries used this column in the WHERE clause. As of now, there are very little records in the vehiclesale table, but as the number of records increase, the index will allow for query times to stay low.

As the data in the database increases, one important way to measure and correct performance would be to enable the slow query log in MySQL. This would allow the database administrator to view slow queries that bottleneck the system and change the physical design to improve performance. For example, as the number vehicles in the system increase, the search function on the website may slow down. The administrator could view which queries are slowing in the log and EXPLAIN those statements. Partitions, table structure, indexes, or table spaces could be added or changed in order to improve performance.

Another obvious problem with the system that could provide a bottleneck as the system scales is the programming language used for the backend. Python is a dynamically typed language, which are generally much slower than statically typed languages. This is a simplified view of the benefits of dynamical and static typing, but using Python causes SQL calls and HTTP responses to be slower than a language like C# or Java. If this system were to be used in a real business, I may consider rewriting the program in a different language.

Member Contributions

The system was created solely by me, Carter Williams. I was responsible for creating the EER diagram, implementing all parts of the system, writing the report, and creating the presentation.

References

[0] “Django documentation | Django documentation | Django,” docs.djangoproject.com. https://docs.djangoproject.com/en/4.1/

[1] zyBooks, a Wiley brand. (2022). zyBooks page, <https://learn.zybooks.com/zybook/WKUCS443YangFall2022/chapter/5/section/7> (accessed 2022).