UNIVERSITY OF PLYMOUTH

School of Engineering, Computing and Mathematics

Student Names:

- Oluwabunmi Oluwayemisi Akintunde
- Chinaka Blessing Ngozi
- Stephen Olanrewaju Bamidele

Student IDS:

- 10938375
- 10940973
- 10938490

Module Title:

Software Development and Databases

Module Code:

COMP5000

# Table of Contents

# List of Figures

## 1. ENTITY-RELATIONSHIP DIAGRAM (ERD) AND THE CHOICE OF DESIGN

An entity-relationship (ER) diagram is the visual representation of a database structure, entities, attributes, and relationships. The first step taken in designing the database was to identify the primary and foreign keys in each table. The primary keys are unique identifiers assigned to each table, while the foreign keys are essentially primary keys from one table referenced in another table.

Afterwards the relationships between the tables were defined. This is an important part of the ERD design. The relationship takes the form of one-to-many and many-to-one relationships. For example, each customer can place multiple orders, which represents a one-to-many relationship. Likewise, many orders might be associated with a single vendor, illustrating a many-to-one relationship.

The database was also normalized using the third normal form (3NF) to minimize redundancy and maintain data integrity by ensuring that all the non-prime attributes depend only on the primary key and not on other non-prime attributes.

The MySQL Workbench was used as the tool in designing this ERD. Tables were created with primary keys, foreign keys, and attributes, and then a reverse engineering technique was used to generate the diagram. While Diagram.net is a simple and versatile design tool, MySQL Workbench was used as the preferred choice because it automatically generated the diagram from the schema, thereby avoiding manual errors and ensuring consistency between the design and implementation, which saves time rather than the dragging and dropping of shapes to represent entities and relationships with Diagram.net. MySQL Workbench also automatically visualizes using Crow's Foot Notation to show the relationships between entities.

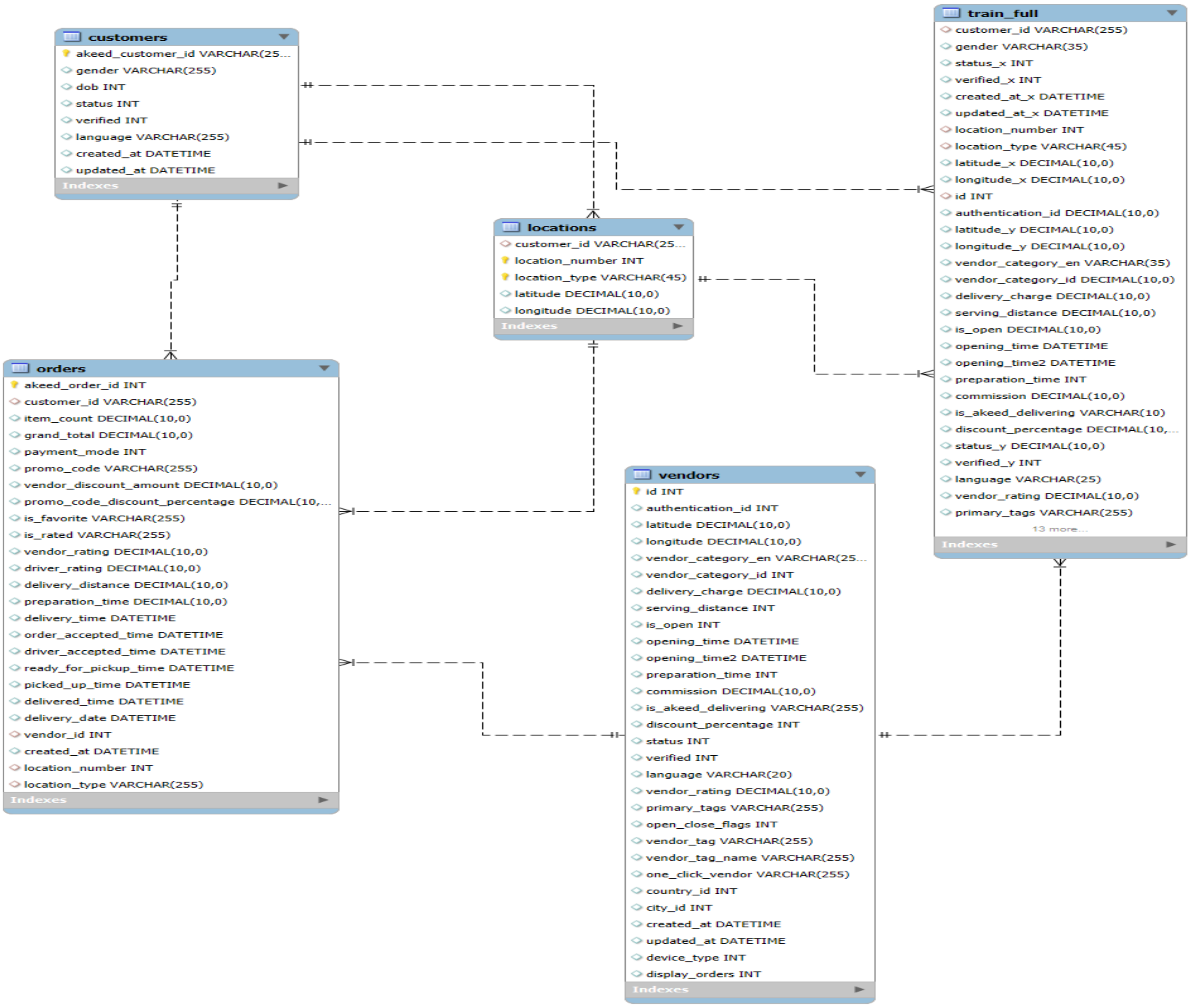


*Figure 1: Entity Relationship Diagram*

**2. THE SQL SCHEMA**

```
-- Creating the Customers table

 CREATE TABLE Customers (

     akeed_customer_id VARCHAR (255) PRIMARY KEY, -- Automatically generates unique IDs

     gender VARCHAR(255),

         dob INTEGER,

         status INTEGER,

         verified INTEGER,

         language VARCHAR(255),

         created_at DATETIME,

         updated_at DATETIME

 );


 -- Creating the Vendors table

 CREATE TABLE Vendors (

     id INT AUTO_INCREMENT PRIMARY KEY,

     authentication_id INTEGER,

         latitude NUMERIC,

         longitude NUMERIC,

         vendor_category_en VARCHAR(255),

         vendor_category_id INTEGER,

         delivery_charge NUMERIC,

         serving_distance INTEGER,

         is_open INTEGER,

         opening_time DATETIME,

         opening_time2 DATETIME,

         preparation_time INTEGER,
    commission NUMERIC,
    is_akeed_delivering VARCHAR(255),
    discount_percentage INTEGER,
    status INTEGER,
    verified INTEGER,
    language VARCHAR(20),
    vendor_rating NUMERIC,
    primary_tags VARCHAR(255),
    open_close_flags INTEGER,
    vendor_tag VARCHAR(255),
    vendor_tag_name VARCHAR(255),
    one_click_vendor VARCHAR(255),
    country_id INTEGER,
    city_id INTEGER,
    created_at DATETIME,
    updated_at DATETIME,
    device_type INTEGER,
    display_orders INTEGER
);


-- Creating the Locations table
CREATE TABLE Locations (
  customer_id VARCHAR (255),
    location_number INTEGER NOT NULL,
    location_type VARCHAR(45) NOT NULL,
    latitude NUMERIC,
        longitude NUMERIC,
        PRIMARY KEY (location_number, location_type), -- Composite Key
          FOREIGN KEY (customer_id) REFERENCES Customers(akeed_customer_id)
);


-- Creating the Orders table
CREATE TABLE Orders (
    akeed_order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id VARCHAR (255),
        item_count NUMERIC,
        grand_total NUMERIC,
        payment_mode INTEGER,
        promo_code VARCHAR(255),
        vendor_discount_amount NUMERIC,
        promo_code_discount_percentage NUMERIC,
        is_favorite VARCHAR(255),
        is_rated VARCHAR(255),
        vendor_rating NUMERIC,
        driver_rating NUMERIC,
        delivery_distance NUMERIC,
        preparation_time NUMERIC,
        delivery_time DATETIME,
        order_accepted_time DATETIME,
        driver_accepted_time DATETIME,
        ready_for_pickup_time DATETIME,
        picked_up_time DATETIME,

        delivered_time DATETIME,

        delivery_date DATETIME,

        vendor_id INTEGER,

        created_at DATETIME,

        location_number INTEGER,

        location_type VARCHAR(255),

        FOREIGN KEY (customer_id) REFERENCES Customers(akeed_customer_id),

        FOREIGN KEY (location_number, location_type) REFERENCES Locations(location_number, location_type),

        FOREIGN KEY (vendor_id) REFERENCES Vendors(id)

);


-- Creating the Train_Full table

CREATE TABLE Train_Full (

    customer_id VARCHAR (255),

        gender VARCHAR(35),

        status_x INTEGER,

        verified_x INTEGER,

        created_at_x DATETIME,

        updated_at_x DATETIME,

        location_number INTEGER,

        location_type VARCHAR(45),

        latitude_x NUMERIC,

        longitude_x NUMERIC,

        id INTEGER,

        authentication_id NUMERIC,
        latitude_y NUMERIC,
        longitude_y NUMERIC,
        vendor_category_en VARCHAR(35),
        vendor_category_id NUMERIC,
        delivery_charge NUMERIC,
        serving_distance NUMERIC,
        is_open NUMERIC,
        opening_time DATETIME,
        opening_time2 DATETIME,
        preparation_time INTEGER,
        commission NUMERIC,
        is_akeed_delivering VARCHAR(10),
        discount_percentage NUMERIC,
        status_y NUMERIC,
        verified_y INTEGER,
        language VARCHAR(25),
        vendor_rating NUMERIC,
        primary_tags VARCHAR(255),
        open_close_flags NUMERIC,
        vendor_tag VARCHAR(255),
        vendor_tag_name VARCHAR(255),
        one_click_vendor VARCHAR(255),
        country_id NUMERIC,
        city_id NUMERIC,
        created_at_y DATETIME,
        updated_at_y DATETIME,
        device_type INTEGER,

        display_orders INTEGER,

        location_number_obj INTEGER,

        id_obj INTEGER,

        target INTEGER,

    FOREIGN KEY (customer_id) REFERENCES Customers(akeed_customer_id),

    FOREIGN KEY (location_number, location_type) REFERENCES Locations(location_number, location_type),

    FOREIGN KEY (id) REFERENCES Vendors(id)

);
```

## 3. DISCUSSION ON DATA CLEANING PROCESS BEFORE LOADING INTO THE DATABASE

The CSV files were loaded into Jupyter Notebook, with the train_full file loaded in chunks because of its large size to allow manageable data processing for inspection of data types and an overview of the contents in the files before proceeding with cleaning. This was done to have a clear understanding of the structure and characteristics of the datasets.

The columns specified in the coursework overview to be ignored were from the vendors and train_full datasets. These columns were considered irrelevant to the objectives of the coursework and were dropped to streamline the database and focus on meaningful data attributes. To make the cleaning process applicable across the files, all datasets were put in a list and assigned to a variable called **datasets.** This allowed a systemic (looping) inspection to check for consistency and missing values across the datasets at once.

### 3.1 Handling Missing Values: Classification of Columns

The columns within each dataset were classified into critical and noncritical categories:

**a)** Critical columns: These included primary and foreign keys, which are necessary for establishing the relationship between the datasets. Missing values in these columns were dropped because such values would comprise data integrity and lead to inconsistencies in the database structure. The order dataset has missing values in its critical columns, particularly the akeed_order_id. Likewise, the location and train_full datasets have missing values in their critical columns. These missing values were removed to maintain the validity of relational links.

b) Non-critical columns: These included columns that were not essential for establishing relationships but added background information, mainly the remaining columns.

### 3.2 Handling Missing values in Non-Critical Columns

A threshold of 50% was set for dropping columns in the non-critical category. Columns with more than 50% missing values were considered to lack sufficient data for meaning analysis and were removed. This decision was made to improve database efficiency and avoid storing sparsely populated data.

An **exception** was made for the DOB column in the customer dataset, which had more than 50% missing values. Despite its high proportion of missing data, the DOB column was retained because it will be needed during this project to add new customers records to the database. Instead of dropping the column, the missing values were filled with 0 to ensure a consistent data type, and the column was converted from float to integer data type.

### 3.3 Handling Missing Values in Other Columns

For numerical columns, missing values were replaced with the **mean** of the respective column to avoid data loss while preserving central tendencies. For categorical columns, missing values were filled with **“Not Specified”** to ensure completeness and maintain a consistent format for categorical data.

### 3.4 Datetime Conversion and invalid Entries

Date columns were converted to the appropriate datetime format for consistency and easier querying in the database. Any invalid entries in date columns were filled with a default timestamp of 00:00:00, ensuring a uniform structure while handling incorrect or incomplete date values.

### 3.5 Foreign Key Consistency Checks

To maintain relational integrity, foreign key columns were checked across datasets for consistency. During this process, unmatched customer IDs were identified in the locations and order datasets. To rectify this, the decision made was to add these unmatched customer IDs to the customers dataset to ensure no data loss while preserving the relationships between datasets.

### 3.6 Final Inspection and Verification

Before loading the datasets into the database, a final inspection was performed to verify the column names, data types, and consistency of the cleaned datasets. This step ensured that the cleaned datasets were ready for seamless integration into the database.

## 4. DEMONSTRATION OF GUI FUNCTIONALITY WITH SCREENSHOTS

A Python-based application with graphical user interface (GUI) was developed to input new customer records into the database using the **tkinter** library. This GUI adds gender, dob, status, verified, language, created_at, and updated_at to the customer's table. Also, the tkinter library was used to provide a simple GUI dashboard so that clicking on the buttons will generate the mean, maximum, and minimum of the 'grand_total' (cost of the order), a histogram of the item_count (using a log scale on the y-axis), and the customer_id of customers that order more than 40 items from the SQLite database. The screenshots of the buttons and evidence that the information about a new customer has been successfully added are shown below.

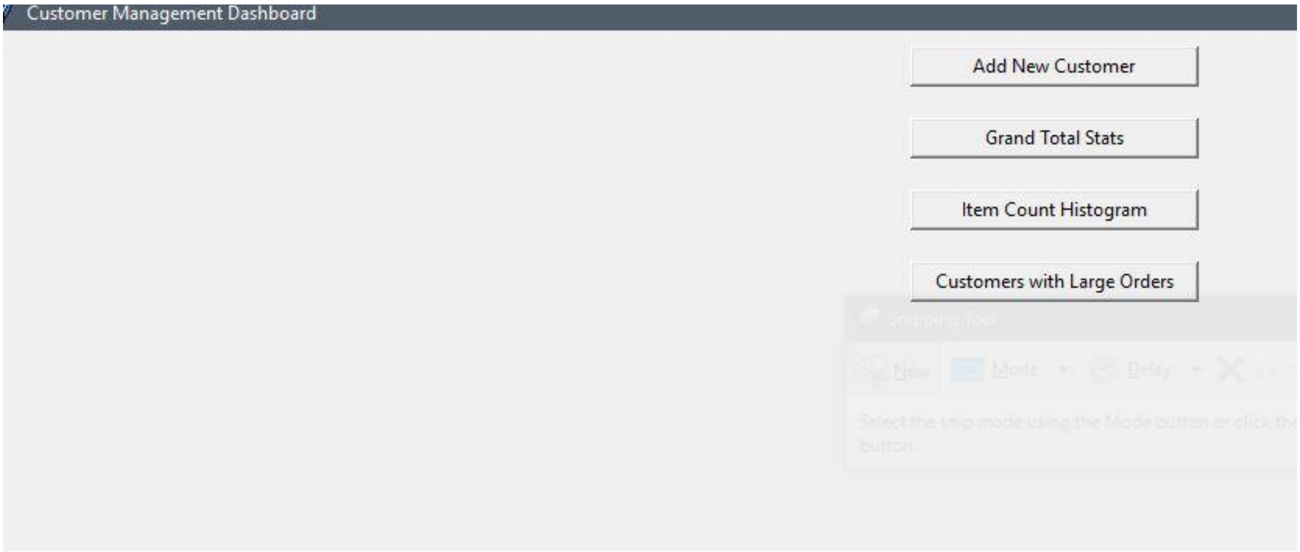


*Figure 2: The GUI Dashboard Buttons*

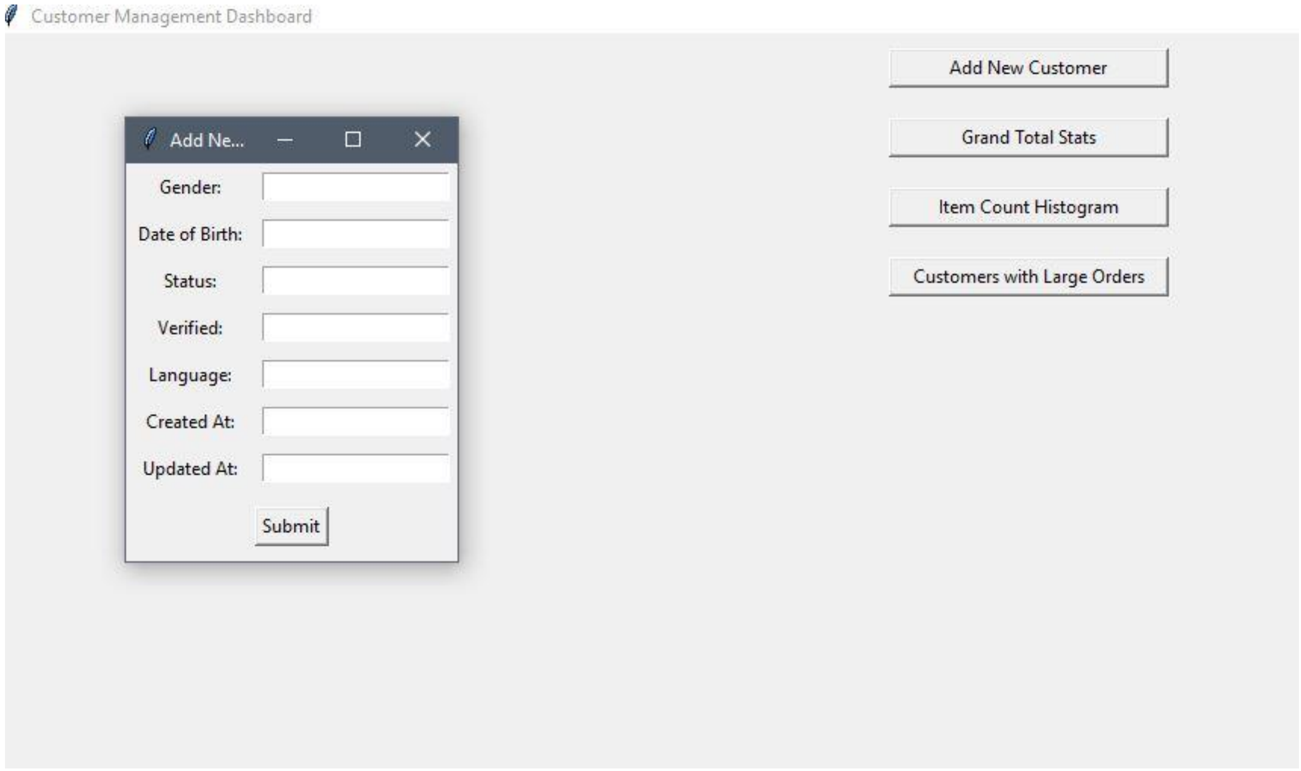
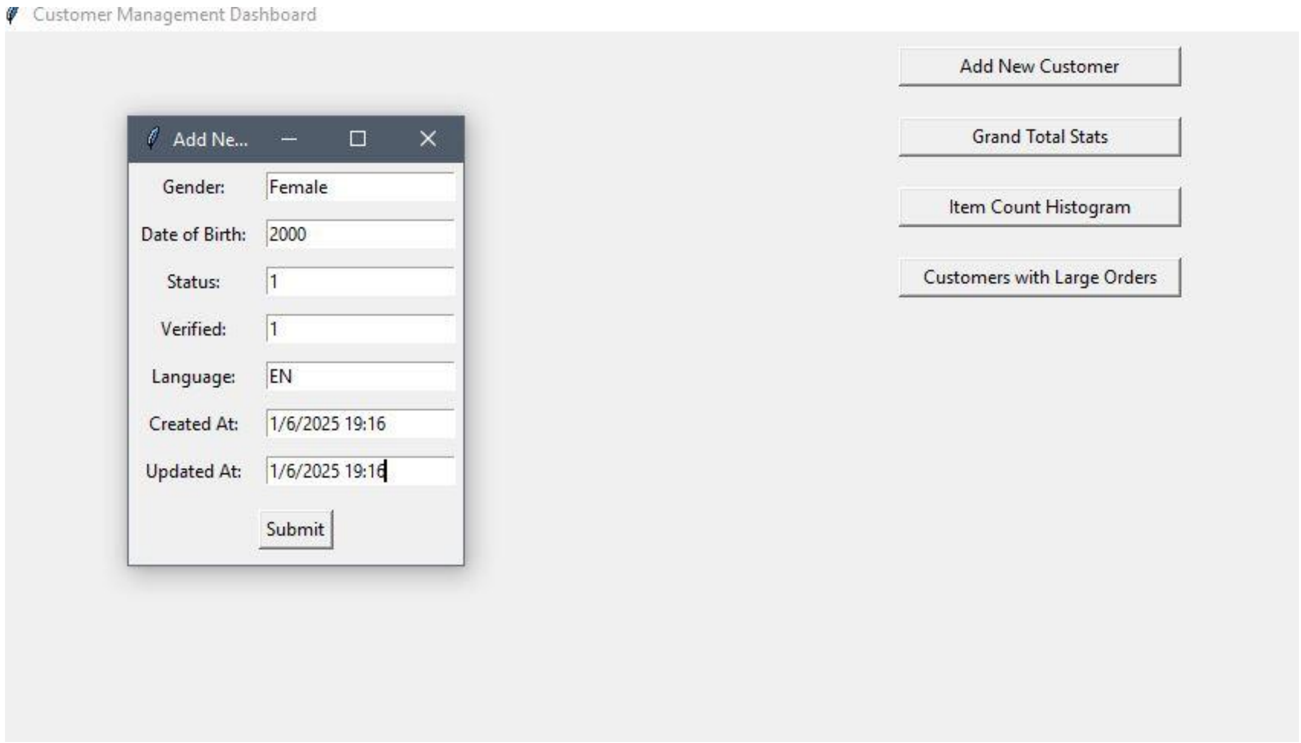
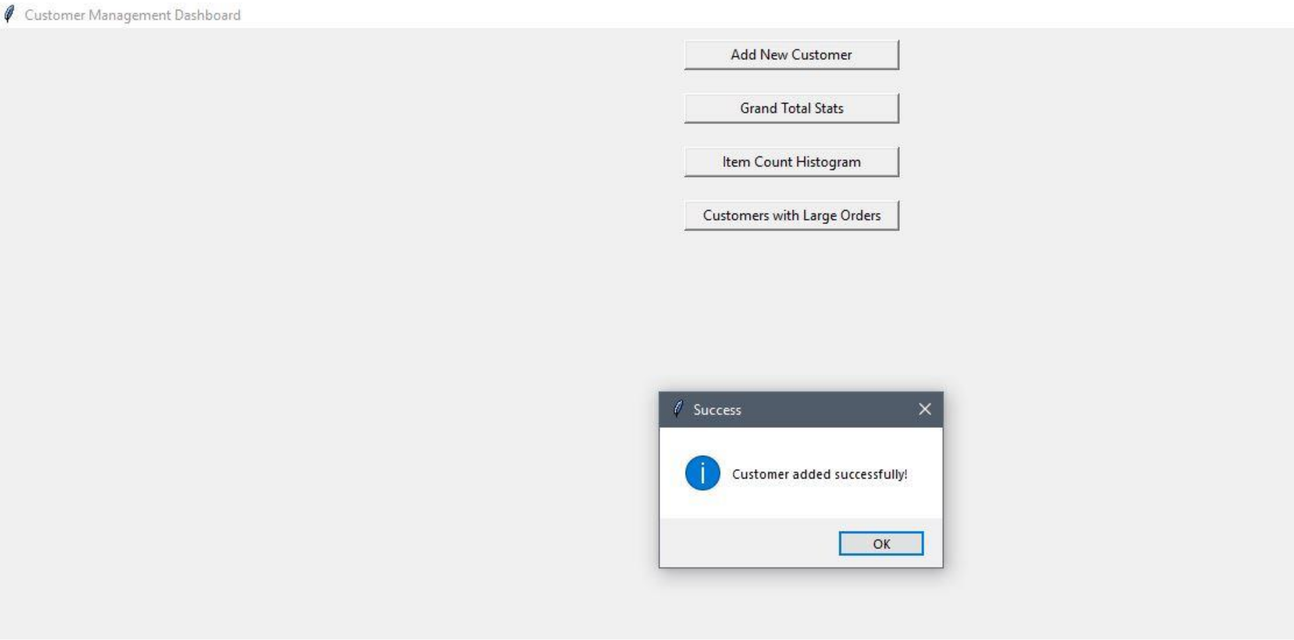


*Figure 3: The GUI to Add a New Customer to the Database*



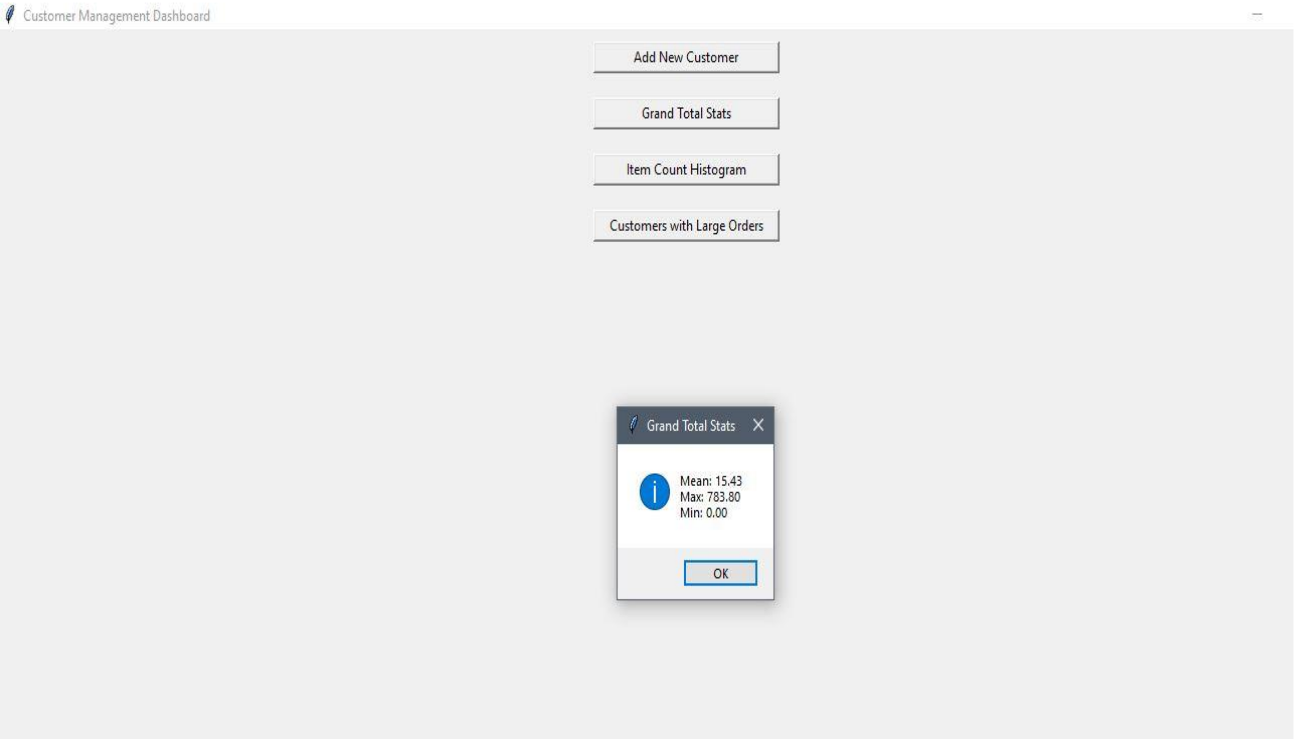
Gender: Female
Date of Birth: 2000
Status: 1
Verified: 1
Language: EN
Created At: 1/6/2025 19:16
Updated At: 1/6/2025 19:16

*Figure 4: New Customer Records*



Customer added successfully!

*Figure 5: New Customer Successfully Added*



```
Mean: 15.43
Max: 783.80
Min: 0.00
```

*Figure 6: The GUI to Calculate the Requested Grand Total Summary Statistics*

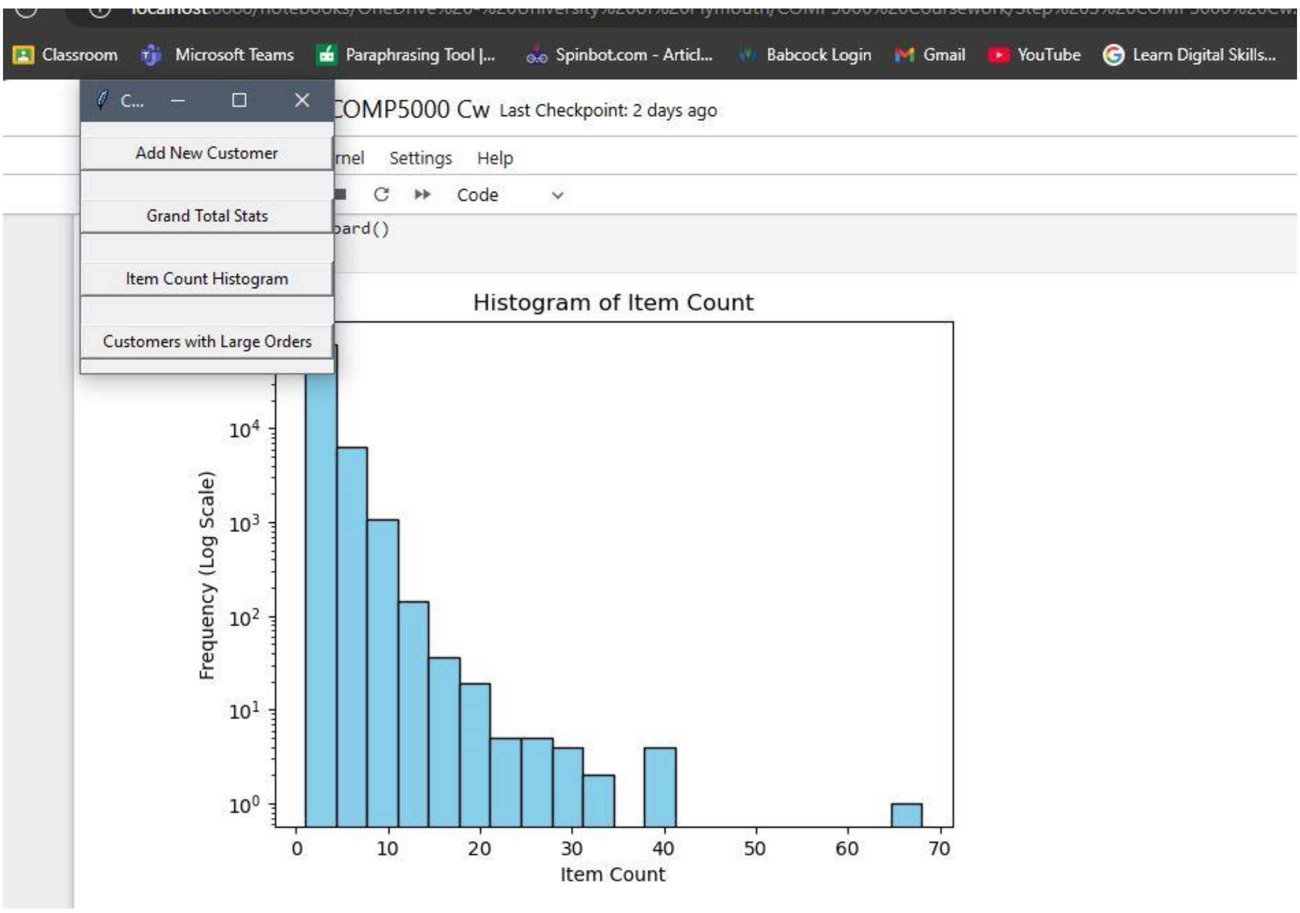
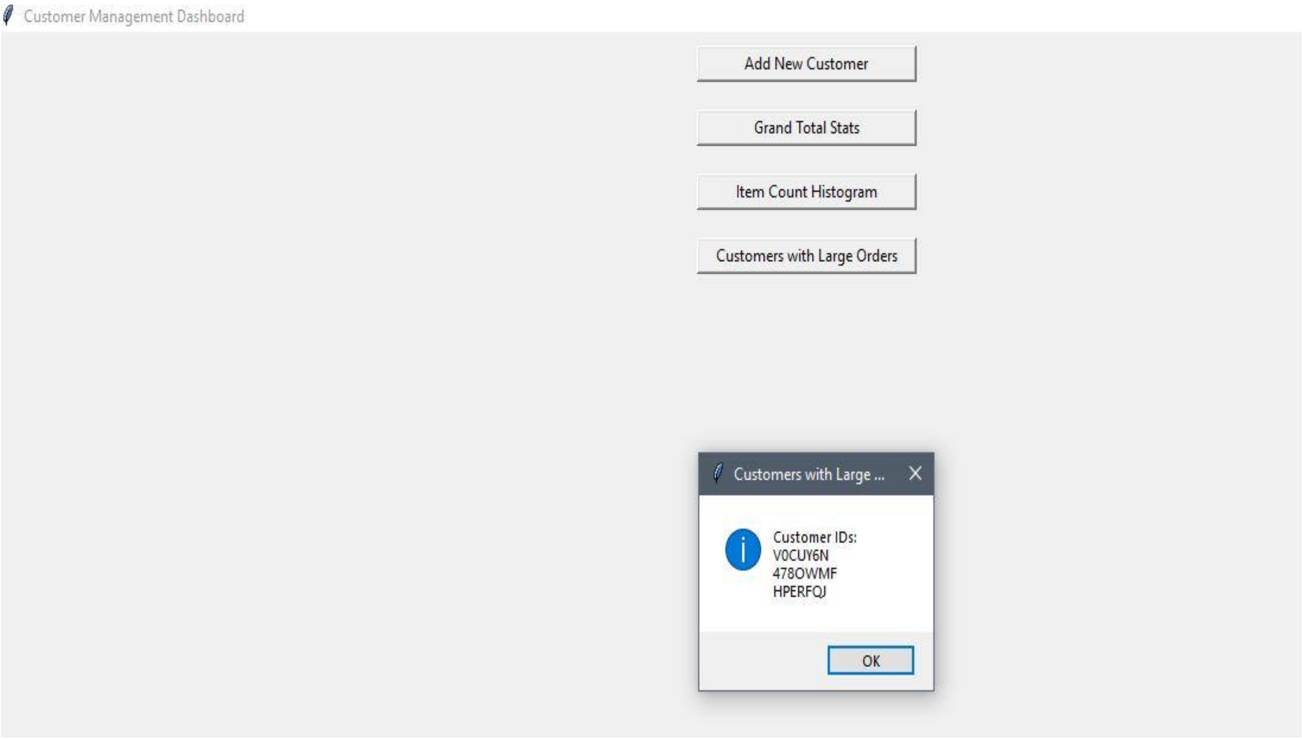


*Figure 7: The GUI which Produces the Histogram of the Item Count*



Customer IDs:
V0CUY6N
478OWMF
HPERFQJ

*Figure 8: The GUI showing Customer IDs of Customers that order more than 40 Items*

# 5. REPORT ON USING AI TO WRITE CODE

## 5.1 TITLE: AI and Coding

Artificial intelligence (AI) is fast transforming software development with tools such as ChatGPT and Google's Gemini. Such tools accelerate programming speed in writing and documenting Python code. A large language model trained on many codebases generates conceptualized code snippets that offer developers a suitable time-saving alternative for implementing solutions. For example, an AI tool could generate the sorting code of a function at once with comments explaining the logic for the developer needing to sort a list of dictionaries by a specific key. It generates a whole bunch of codes instantly. Beyond code generation, the AI tools serve as effectively as ones dedicated to the I/O processing of documentation for complex codebases. Python supports docstring according to PEP 256 guidelines, along with inline comments and comments on functions. Thus, such tools help developers to focus on high-level design decisions rather than wasting time on boilerplate chores. AI coding has been proved to save up to 30% of development time (Nguyen et al., 2024). However, while it is important to enjoy all these benefits, it is equally important to understand that AI-generated code is not always error-free, thus warranting further testing and validation prior to deployment.

An extremely important aspect of the code generated independently by AI is that it needs to be tested to ensure its reliability and correctness. It generates code that may be valid and logically sound in syntax but not cater to any specific cases or requirements of the application. For example, an AI would not consider certain security holes when working with user data/input-created Python function. Unit tests, integration tests, and performance testing must all be done to ensure that the generated code will behave as one intended under any conditions. Pytest can automate this. Most importantly, there arrive occasions when AI introduces very minor bugs or inefficiencies thereby needing a thorough review from the developer of generated code. AI tools also propose practices unsafe among coding, such as not using sensitive data properly (Shneiderman, 2020). Thereby, static code analysis tools like Bandit or SonarQube, should be used for the detection of any possibility of security flaws. All these activities can be brought together into achieving fast development using AI tools without compromising code quality since they all merged with automated tests and human oversight.

Apart from the technical issues, AI generated code raises security and ethical issues. One of the most troubling aspects is the introduction of vulnerabilities into applications. Since AI models train based on every bit of publicly available code, it is possible to generate code that embodies known vulnerabilities, for example, an AI tool generating a function that employs a deprecated encryption algorithm, which makes the application potentially expose to attacks. The ethical stakes in terms of intellectual property and copyright come up here. An AI model may accidentally reproduce proprietary code that is in its corpus, which can subject the potential litigations. Complementary with actions to adopting guidelines on acceptable use of AI-generated code, organizations must provide for strict code review and adherence to open-source licensing norms. The advancements of AI themselves should be open as well. If at all arguable by concerned parties, this will maintain system accountability, first by requiring developers to declare when and where AI technologies are employed, embedded, or invoked in the codebase. AI testing procedures, security issues, as well as ethical standards should be prioritized over efficiency owing to the nature of the technology such as ChatGPT and Gemini in improving Python programming. Hence, it should be utilized responsibly in an efficient manner.

## 5.2 References

Nguyen, N.T., Pham, H.P., Cao, C.M. and Hoang, A.D., The Development of Research on AI and Cost Saving from 1990 to 2024: A Bibliometrics Review from the Web of Science Database.

Shneiderman, B., 2020. Bridging the gap between ethics and practice: guidelines for reliable, safe, and trustworthy human-centered AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *10*(4), pp.1-31.