DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

FarMarT Business Operations

Matthew Schlatter

4/21/2023

8.0

This document outlines a sales subsystem for a farming company, FarMarT. This subsystem tracks sales, organizes data, and calculates totals based on the data.

Revision History

This table documents the various major changes to this document.

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Matthew Schlatter	2023/02/17
2.0	Updated to be more technical	Matthew Schlatter	2023/02/24
3.0	Updated UML chart and added transition to the chart	Matthew Schlatter	2023/03/03
4.0	Added an updated UML chart, ER diagram and a description of the SQL database	Matthew Schlatter	2023/03/21
5.0	Updated ER diagram and added database interface details	Matthew Schlatter	2023/04/04
6.0	Added descriptions for sections 3.3 and 3.4	Matthew Schlatter	2023/04/06
7.0	Updated UML chart	Matthew Schlatter	2023/04/07
8.0	Moved location of business operations, added abbreviation, and overall, more elaboration in each section	Matthew Schlatter	2023/04/21

Contents

1. Introduction

This design creates a sales subsystem for the family-owned business FarMarT (FMT), a regional chain of Business-to-Business (B2B) stores that sell farm equipment and supplies to farmers large and small, such as seeds, tractors, soil, etc. With the recent acquisition by Pablo Myers, a revamp of this system is favorable to increase profits. This entire system is independently developed, and the sales subsystem efficiently gathers and sorts valuable customer, order, store, and invoice data as well as calculating totals for orders.

1.1 Purpose of this Document

This document is designed to document the process of creating this subsystem. It lays an outline for the entire project and explains chosen design decisions, as well as pros and cons of many design options.

1.2 Scope of the Project

This project is designed to be utilized by FarMarT, specifically following their business model and procedures. It is designed to make orders easily accessible and trackable. This subsystem efficiently collects data and stores it in a database using Electronic Data Interchange where it can be gathered to create reports. This system does not deal with customer issues or marketing, only the tracking of sales. Totals are calculated with any applicable taxes plus the price of the customer's desired item and amount. For example, if the customer would like 10 bags of soil, at $10/bag, this would cost $107.15 with tax because products have a tax of 7.15%. Furthermore, services have a tax of 3.45%, purchased equipment or leases less than $10000 have no tax, leases $10000-$100000 have a $500 tax, and leases >= $100000 have a tax of $1500.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

All necessary definitions for this document can be found here.

None

1.3.2 Abbreviations & Acronyms

All necessary abbreviations and acronyms for this document can be found here.

FMT – FarMarT (Company Name)

B2B – Business-to-business

2. Overall Design Description

This project takes in many different data points and provides different data files based on the provided information. Data files include invoices with customer orders and order information, customer data including name and contact information, as well as store information where the order was placed. Person, store, and item data is imported through CSV files and exported as XML and/or JSON files.

Invoice data is printed to the standard output in the format of a receipt. Additionally, an SQL database is implemented to make the management of mass amount of data from these orders easier.

2.1 Alternative Design Options

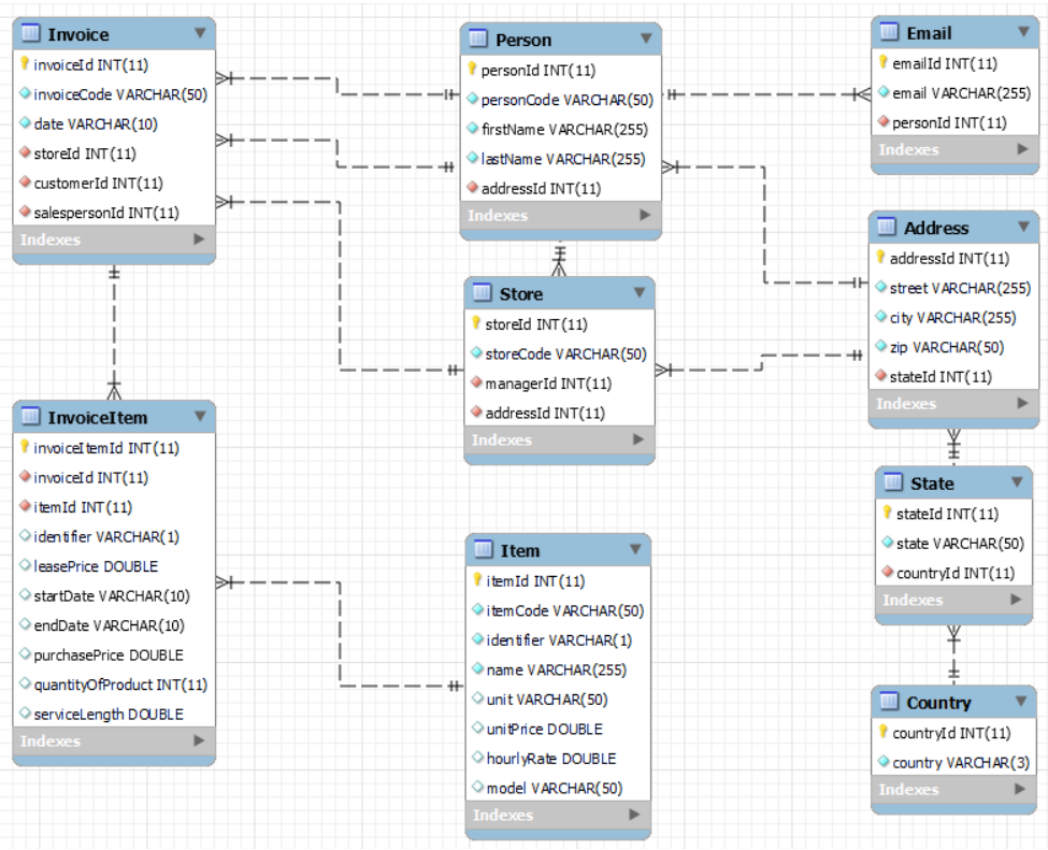
Loading and converting data can be placed into a separate package to allow further separation of operations. Additionally, interfaces can be implemented to increase inheritance and reduce issues with abstract classes.

3. Detailed Component Description

This design is based in Java with a connection to an SQL database. Data is stored in a database and is retrieved and persisted using JDBC queries.

3.1 Database Design

This database utilizes 11 tables to store data of different data types. Primary keys are denoted by a *, non-primary keys are denoted by a +, and foreign keys are denoted by a -. All foreign keys directly correlate to the key in the subsequent table and no value can be null. By utilizing a database, it will be simpler to deal with the massive amount of input data that will be received with orders as well as data corresponding to products and services. Data is normalized by creating a state and country table for the address. It is common to have customers from the same state or country, but not from the same zip, city, or street.



3.1.1 Component Testing Strategy

Data is inserted into the database and tested using multiple different queries to ensure the desired results are output. Queries include retrieving basic person or order information, adding emails to a record, finding specific item information, and total sales by employees or stores.

3.2 Class/Entity Model

Outline of classes and their specific variables and methods. There are three main sections. The first deals with all basic data types, the second deals with loading, converting, and outputting the data, and the last deals with invoices and invoice items.

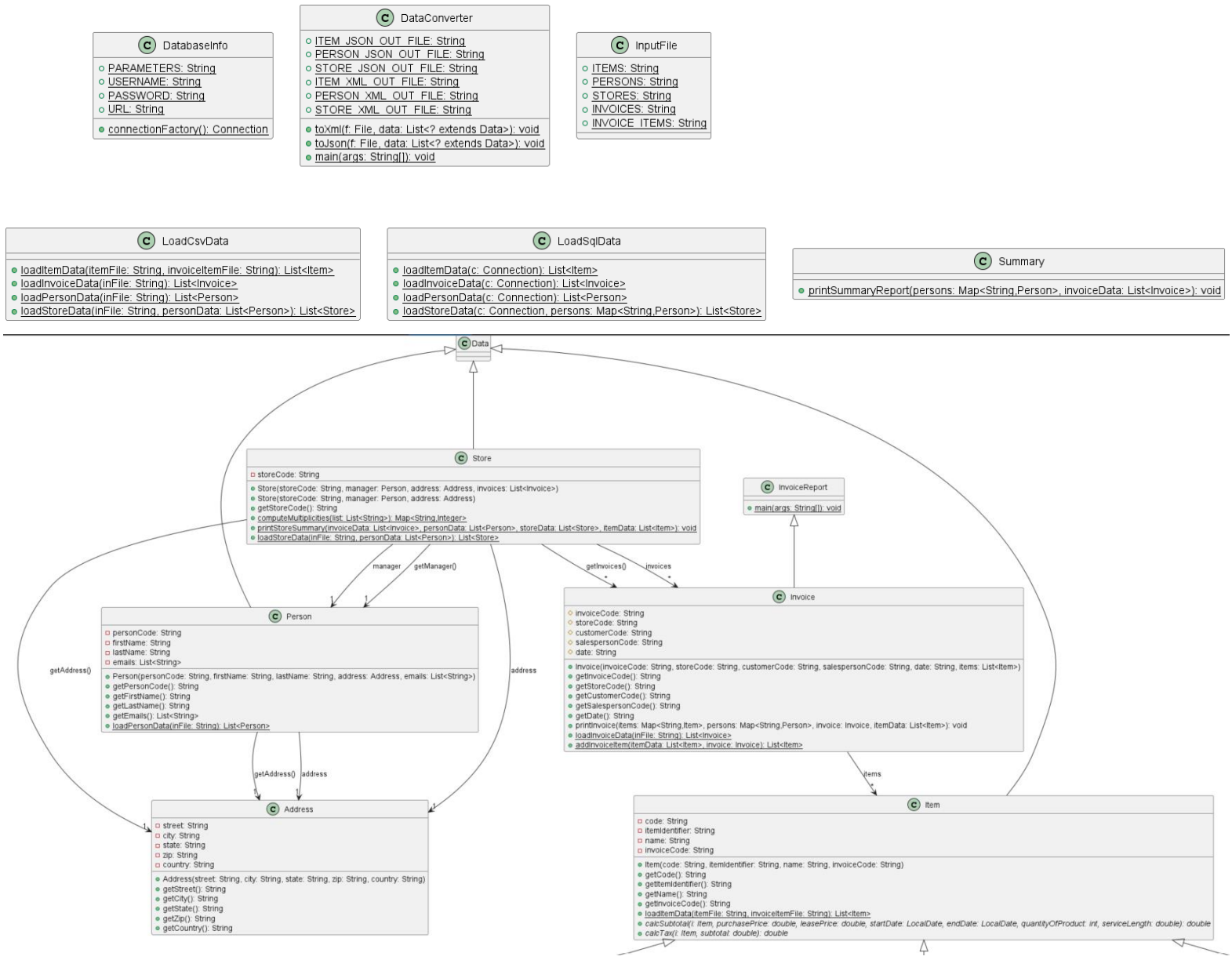


Figure 1: UML Diagram for the FarMarT operations, outlining different classes and variables.

3.2.1 Component Testing Strategy

Test files with randomly generated data from Mockaroo were input into the classes above to test the effectiveness of scanning and printing data. Data was imported through CSV files and exported to XML/JSON files. Each class passed their respective tests.

Additionally, test files for invoices were generated with Mockaroo data as well as utilizing the item, person, and store data. An output.txt file shows expected output of invoices in a receipt format. Totals were calculated by hand adhering to the business logic outlined in the introduction of this document.

3.3 Database Interface

JDBC is used in the Java application to connect with an SQL database, allowing input directly from the database using queries as opposed to scanning CSV files. A factory function was created to establish a connection to the database, and each Java class was appropriately mapped to each SQL table. Wildcards are avoided in select statements to optimize query performance and reduce bandwidth usage.

3.3.1 Component Testing Strategy

Various select statements were used to take data from the SQL database and seamlessly turn into Java objects from ResultSets due to a one-to-one relationship. The results were compared with SQL

statements in MySQL Workbench to ensure the same data was accurately selected and all the data that was desired was received.

3.4 Design & Integration of Data Structures

A Java class is used to persist data into the database in case additional data needs to be added. This is done by using various insert queries and update PreparedStatements. Additionally, the generated keys are returned and stored so other classes that require them as a foreign key can use them. This is done using Statement.RETURN_GENERATED_KEYS and ResultSets.

3.4.1 Component Testing Strategy

Similarly, to the select statements, various insert and delete statements were designed to test. To verify the effectiveness of said statements, select statements were run in MySQL Workbench to check that the specified data was inserted/deleted.

3.5 Changes & Refactoring

While this subsystem was created, a major redesign had to take place to follow more object-oriented principles. This included the removal of unnecessary classes and incorporation of their functionality into other, more reasonable places. For example, invoice items were originally given a class each. However, this resulted in repeated information and information being stored in different areas for the same item.