DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

FarMarT Business Operations

Design Document

Matthew Schlatter

5/12/2023

9.0

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

Contents

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
9.0	Elaborated on 3.4, 3.1.1, 2, 2.1, moved tax information to 1.3.1, and overall minor improvements	Matthew Schlatter	2023/05/12

1. Introduction

This design document outlines 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 the sales subsystem is favorable to increase efficiency. The 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 the creation of the subsystem. It lays an outline for the entire project and explains chosen design decisions, pros and cons of many design options, and how each component of this design was tested.

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 an Electronic Data Interchange format 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. Refer to the 1.3.1 Definitions for tax rates.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

All necessary definitions for this document can be found here.

Product tax – 7.15%

Service tax – 3.45%

Equipment tax – $0 if purchased or leased under $5000, $500 if leased at or above $10000 and below $100000, $1500 if leased at or above $100000.

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

JDBC – Java EE Database Connectivity

2. Overall Design Description

This project takes in many different data points and outputs are based on the provided information. Data includes invoices with customer orders and order information, customer data including name and contact information, as well as store information where the order was placed. All data is imported through CSV files and/or a SQL database and exported as XML and JSON files. Additionally, invoice data can be printed to the standard output in the format of a receipt. The SQL database was implemented to manage large amounts of data with ease. This database is connected to the Java implementation using JDBC.

2.1 Alternative Design Options

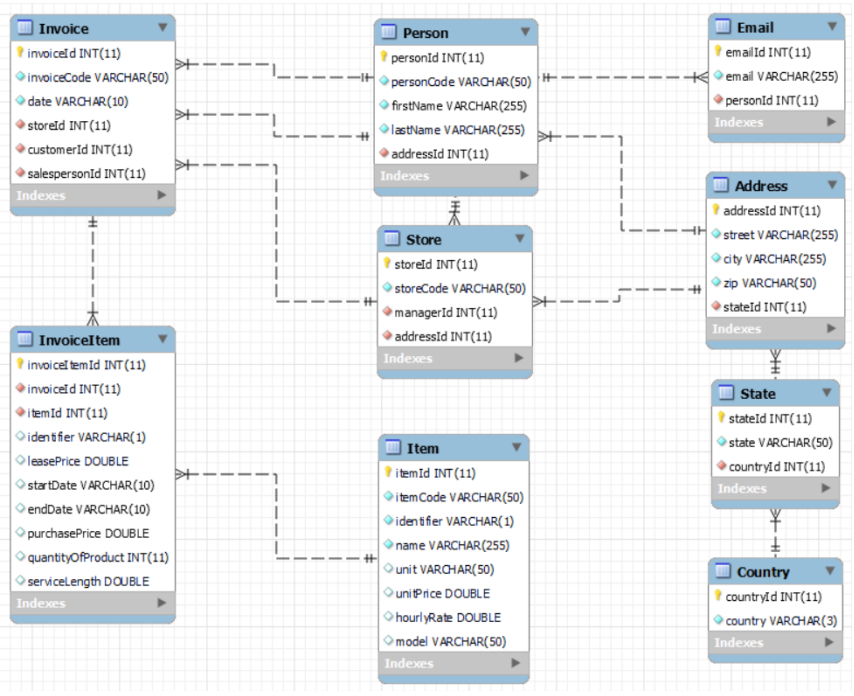
Alternative designs include placing the loading and converting data functions into a separate Java package to allow further separation of operations. Additionally, interfaces can be implemented around the Item class to strengthen inheritance and reduce issues with abstract classes, such as calcSubtotal and calcTax.

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 9 tables to store data of different data types. All foreign keys directly correlate to the key in the subsequent table as directed by the lines and arrows. 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. Furthermore, foreign keys are used instead of repeating customer or item codes to ensure easy discrimination between datapoints.



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. 12 total queries were made, which include retrieving basic person or order information, adding emails to a record, finding specific item information, and total sales by employees or stores. For example, the query “select p.firstName,p.lastName,e.email from Person p left join Email e on p.personId = e.personId where p.personId = 12;” can be used to ensure that the person with personId 12 has the correct emails associated with their database profile. All tests passed.

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.

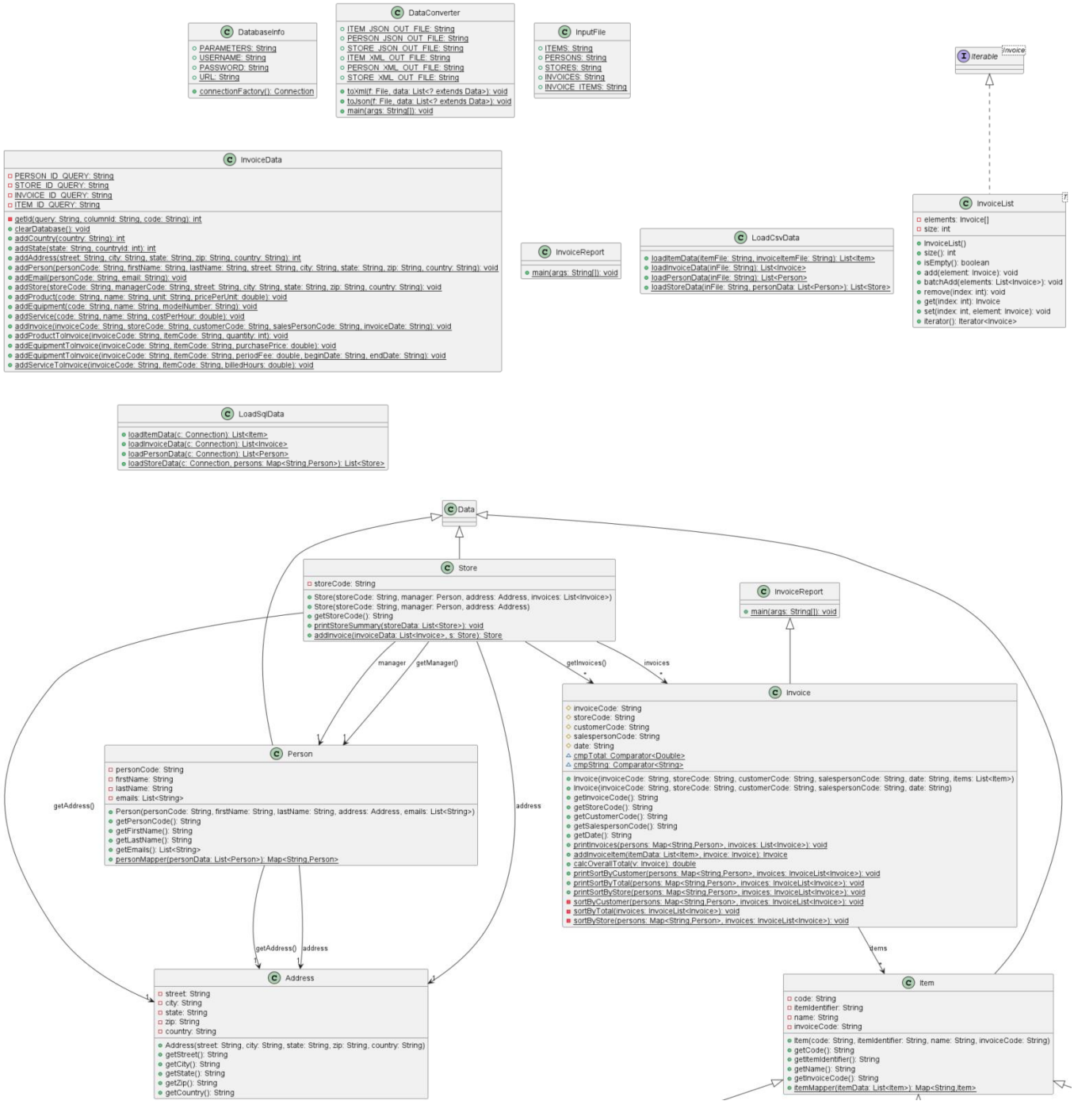


Figure 1: UML Diagram for the FarMarT operations, outlining different classes and variables.

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

Eight total select statements were used to take data from the SQL database and seamlessly turn it into Java objects from ResultSets, some of which were for foreign key ids, and some were for full store or person data. 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 (InvoiceData) is used to persist data into the database for new orders. This is done by using various insert queries and update PreparedStatements. Additionally, the generated keys are returned and stored so other classes can use it as a foreign key, such as the country and state id for address. This is done using Statement.RETURN_GENERATED_KEYS and ResultSets.

3.4.1 Component Testing Strategy

Similarly to when select statements were used to verify data directly in MySQL before using JDBC, various select statements were designed in JDBC to test the data persistence. To verify the effectiveness of said statements, select statements were also run in MySQL Workbench to check that the specified data was inserted/deleted as well as being selected properly by JDBC.

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 locations. 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 which was unreasonable.