Academic year 2024 - 2025

BOVITA

بـــوفيتا

Sales Forecasting System

Sarra Ibn El Haj - Senior BA/IT

Sarra Ibn El Haj - Senior BA/IT

What is BOVITA?

BOVITA is a startup focused on providing 100% natural treatment and prevention solutions for bovine mastitis, with a product line that includes creams and gels. As the CEO, I aim to leverage a Sales Forecasting System to predict sales trends and make informed decisions regarding pricing, inventory, and marketing strategies. This system will help BOVITA streamline operations, optimize resources, and maximize revenues.

Why is this system important?

- **Predicts sales trends** to help BOVITA make data-driven decisions.
- **Improves inventory management** by forecasting demand.
- Helps in **setting realistic revenue goals** based on historical data.

Technologies Used

- Programming Language: Java
- Database: SQLite
- Visualization: JFreeChart
- Frontend: JavaFX
- Build Tool: Maven
- Testing: JUnit 5

Features

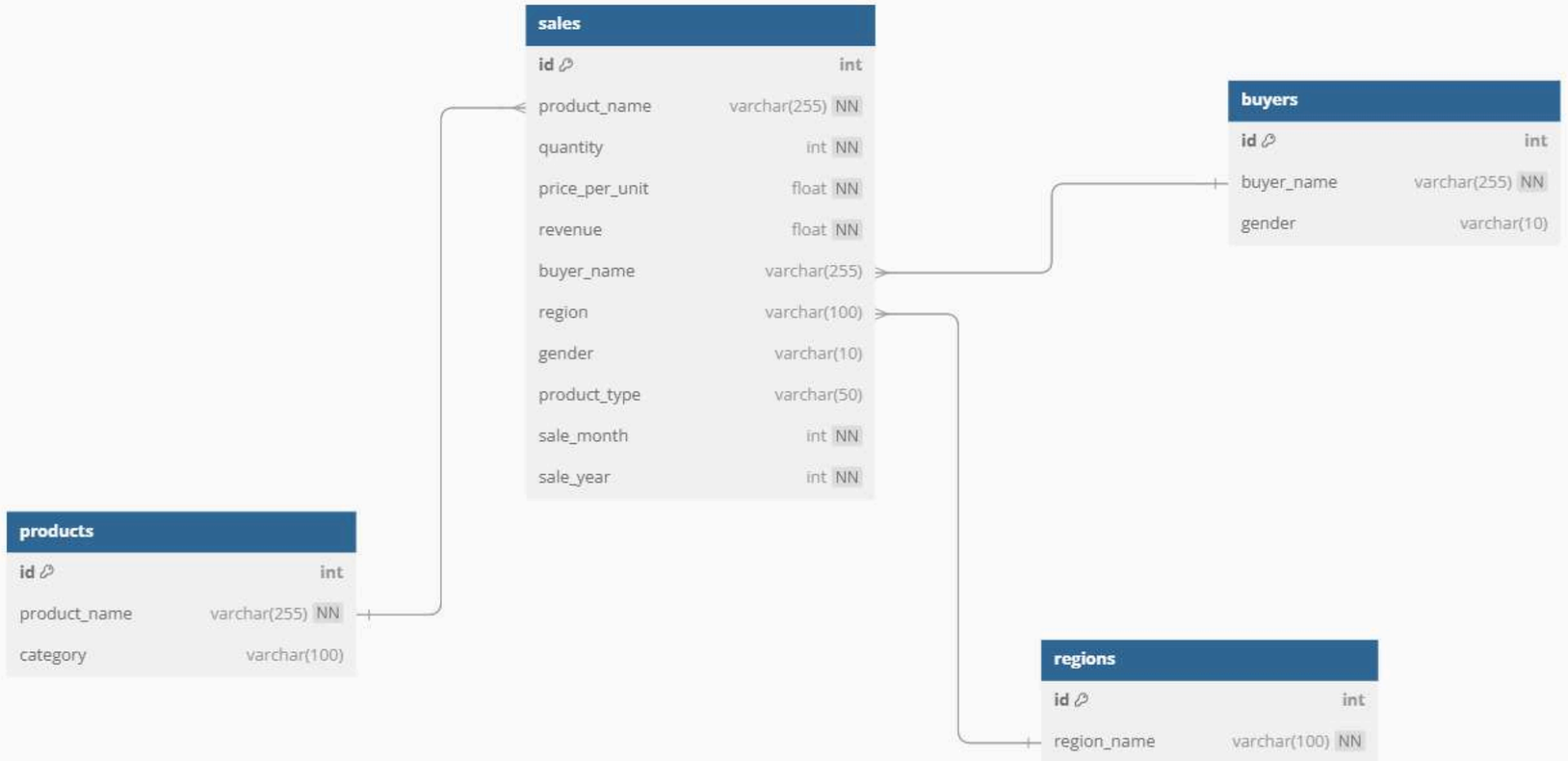
- ✅ Sales Data Management: Add, update, and delete sales records
- ✅ Revenue Calculation: Computes total revenue & revenue by product/region
- ✅ Forecasting System: Uses past trends to predict future revenue
- ✅ Graphical Visualization: Pie charts, line graphs, bar charts
- ✅ Data Persistence: Stores sales data in SQLite
- ✅ User-friendly UI: Built with JavaFX

System Architecture

- User Interface (JavaFX UI) → Allows users to input and visualize sales data
- Database Layer (SQLite) → Stores all sales records
- Business Logic (Java) → Handles revenue calculation & forecasting
- Visualization Layer (JFreeChart) → Generates interactive charts

Sarra Ibn El Haj - Senior BA/IT

Database Schema



Forecasting Model

📌 How does the forecasting system work?

- Uses historical revenue data
- Calculates average growth rate
- Applies a weighted trend model to predict future revenue

📌 Formula Used:

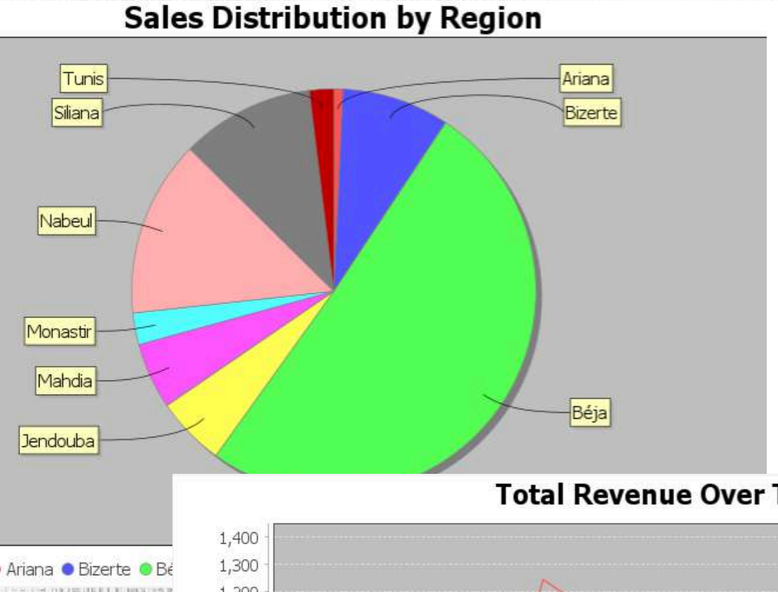
- $FutureRevenue = LastRevenue \times (1 + AverageGrowthRate)$

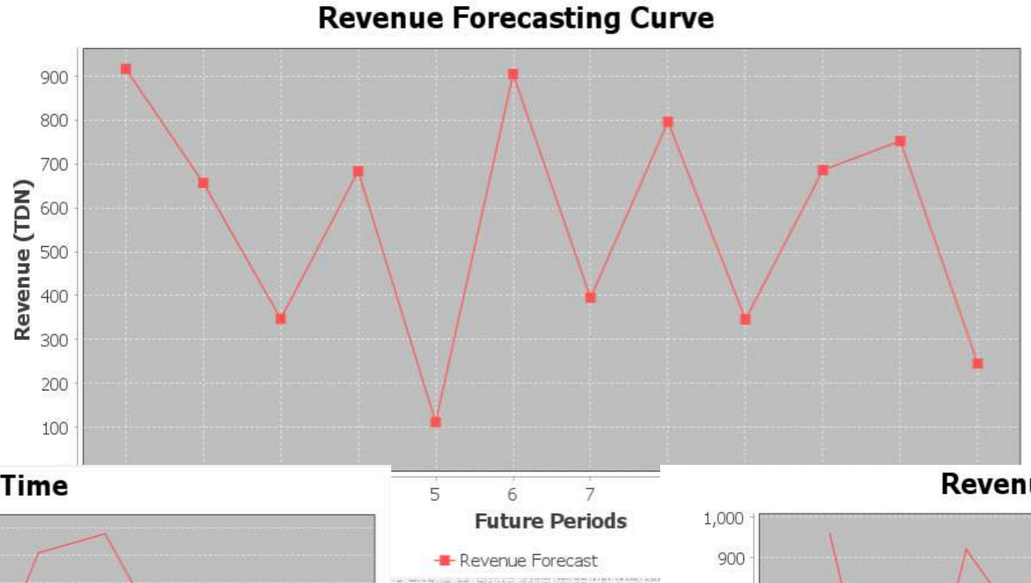
Visualization & Reporting

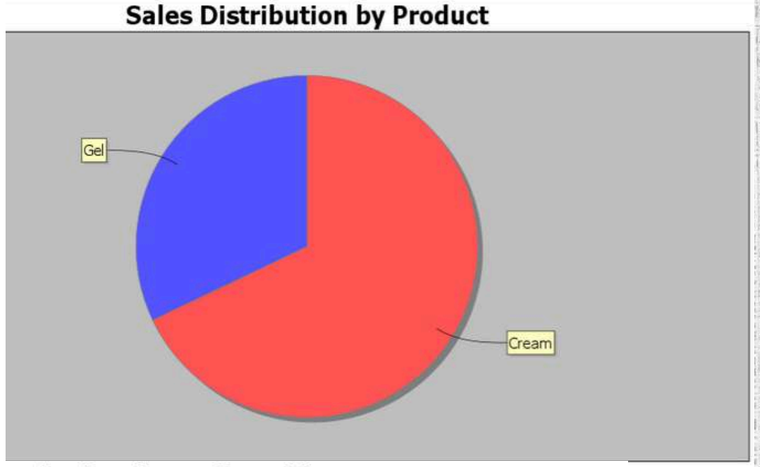
Graph Type	Used For	Why Used?
Line Chart 📈	Total Revenue Over Time	Revenue trends & seasonality
Line Chart 📉	Revenue by Product Over Time	Identifying best-selling products
Bar Chart 📊	Revenue by Region Over Time	Regional sales comparison
Line Chart 🔮	Revenue Forecasting Curve	Future revenue prediction
Pie Chart 📊	Sales Distribution by Product	Product performance comparison
Stacked Bar Chart 📊	Sales Distribution by Region	Regional contribution to sales

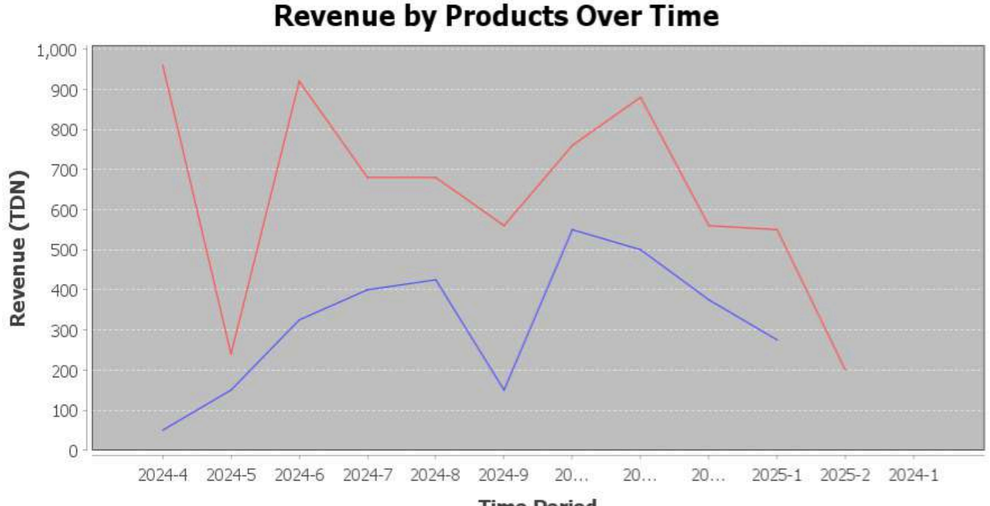
Sarra Ibn El Haj - Senior BA/IT

Visualization & Reporting









How the classes work together?

Class	Depends On	Description
App	DatabaseHandler, VisualizationPanel	Manages UI & user interactions
DatabaseHandler	SalesRecord	Handles database CRUD operations
SalesRecord	-	Represents a sales transaction
VisualizationPanel	DatabaseHandler	Generates charts & insights
AppTest	DatabaseHandler	Ensures correctness of system logic

Sarra Ibn El Haj - Senior BA/IT

Object Oriented Programming Principles Implemented ✅

Encapsulation

Example: SalesRecord Class

- The SalesRecord class encapsulates the sales data, ensuring that fields can only be accessed or modified through getter and setter methods.

✅ Private variables prevent direct modification

✅ Public getters and setters allow controlled access

✅ The getRevenue() method encapsulates revenue calculation within the class

```java
package com.example;

public class SalesRecord {
    private String productName;
    private int quantitySold;
    private double pricePerUnit;
    private String buyerName;
    private String region;
    private String gender;
    private String productType;
    private int saleMonth;
    private int saleYear;
    private double revenue;

    // Constructor
    public SalesRecord(String productName, int quantitySold, double pricePerUnit, St
                       String region, String gender, String productType, int saleMon
        this.productName = productName;
        this.quantitySold = quantitySold;
        this.pricePerUnit = pricePerUnit;
        this.revenue = quantitySold * pricePerUnit;
        this.buyerName = buyerName;
        this.region = region;
        this.gender = gender;
        this.productType = productType;
        validateSaleDate(saleMonth, saleYear);
        this.saleMonth = saleMonth;
        this.saleYear = saleYear;
    }
```

Sarra Ibn El Haj - Senior BA/IT

Abstraction

Example: DatabaseHandler Class

- Interacts with SQLite Database
- Provides high-level methods to store, retrieve, and manipulate sales data without exposing database complexities.

✅ Database complexity is hidden from the UI

✅ High-level methods (like getRevenueByProductOverTime()) expose only essential functionalities

✅ Prevents direct SQL queries inside other classes

```java
public class DatabaseHandler {

    private static final String DB_URL = "jdbc:sqlite:sales.db";

    // Initialize the database and create the "sales" table if it doesn't exist
    public void initializeDatabase() {
        String createTableQuery = """
                CREATE TABLE IF NOT EXISTS sales (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    product_name TEXT NOT NULL,
                    quantity INTEGER NOT NULL,
                    price_per_unit REAL NOT NULL,
                    revenue REAL NOT NULL,
                    buyer_name TEXT,
                    region TEXT,
                    gender TEXT CHECK (gender IN ('Male', 'Female', 'Other')),
                    product_type TEXT CHECK (product_type IN ('Prevention', 'Treatment')),
                    sale_month INTEGER NOT NULL,
                    sale_year INTEGER NOT NULL
                );
                """;
        try (Connection connection = DriverManager.getConnection(DB_URL);
             Statement statement = connection.createStatement()) {
            statement.execute(createTableQuery);
        } catch (SQLException e) {
            System.err.println("Error initializing database: " + e.getMessage());
        }
    }
```

```java
public Map<String, Map<String, Double>> getRevenueByProductOverTime() {
    String query = """
        SELECT product_name,
               sale_year || '-' || sale_month AS period,
               SUM(revenue) AS total_revenue
        FROM sales
        GROUP BY product_name, period
        ORDER BY product_name, sale_year, sale_month;
    """;

    return fetchMultiKeyDataMap(query);
}
```

Inheritance

Example: App class Inherits from Application

✅ App inherits JavaFX's Application class

✅ start() overrides the Application class method

✅ launch(args); starts the JavaFX UI lifecycle

```
public class App extends Application {

    private final DatabaseHandler dbHandler = new DatabaseHandler();

    Run | Debug
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        // Initialize the database (without clearing records)
        dbHandler.initializeDatabase();

        // Create a TabPane for clean UI
        TabPane tabPane = new TabPane();

        // Tabs for Data Entry and Visualization
        Tab dataEntryTab = new Tab("Data Entry", createDataEntryPanel());
        Tab visualizationTab = new Tab("Visualization", new VisualizationPanel(dbHandler).getVisualiz

        // Disable tab closing
        dataEntryTab.setClosable(false);
        visualizationTab.setClosable(false);

        // Add tabs to the TabPane
        tabPane.getTabs().addAll(dataEntryTab, visualizationTab);
```

Polymorphism

Example: getSalesData() Using Polymorphism

The method fetchDataMap is a generic method that can fetch different types of sales data.

✅ Code reuse using a single method (fetchDataMap) for different functionalities

✅ Different behavior depending on the SQL query provided

```java
// Helper function: Fetch data for single-key maps
private Map<String, Double> fetchDataMap(String query) {
    Map<String, Double> dataMap = new LinkedHashMap<>();
    try (Connection connection = DriverManager.getConnection(DB_URL);
         Statement statement = connection.createStatement();
         ResultSet resultSet = statement.executeQuery(query)) {
        while (resultSet.next()) {
            dataMap.put(resultSet.getString(columnIndex:1), resultSet.getDouble(columnIndex:2));
        }
    } catch (SQLException e) {
        System.err.println("Error fetching data: " + e.getMessage());
    }
    return dataMap;
```

Live Demo

Challenges & Solutions

🔴 Challenges Faced:

- Integrating JavaFX with SQLite
- Implementing accurate forecasting
- Handling large datasets efficiently

✅ Solutions Implemented:

- Optimized SQL queries for better performance
- Used JFreeChart for dynamic visualizations
- Applied growth-based forecasting models

Closing Word