Tokopee Inc. Database Overhaul

Full Technical Report

Phase 1–4

Prepared by:
Gregorius Willson - 2802449846
Luis Alexandro Soetherio - 2802425360
Kenrick Yovan - 2802517784
Computer Science, BINUS University

January 8, 2026

Contents

Chapter 1

Introduction

Project assignment for Tokopee Inc. (AOL DB). An online retail shop. For years, their operations relied on a single flat CSV file containing all historical transactions. This structure created performance issues, redundancy, and inconsistency.

This project replaces the old flat-file system with a fully normalized relational database following best practices.

This report covers:

- Database analysis and 3NF ERD design
- SQL schema implementation (DDL)
- Data manipulation and analytical queries (DML)
- Advanced automation (Triggers and Stored Procedures)

Chapter 2

Phase 1: ERD (3NF)

2.1 Entities Identified

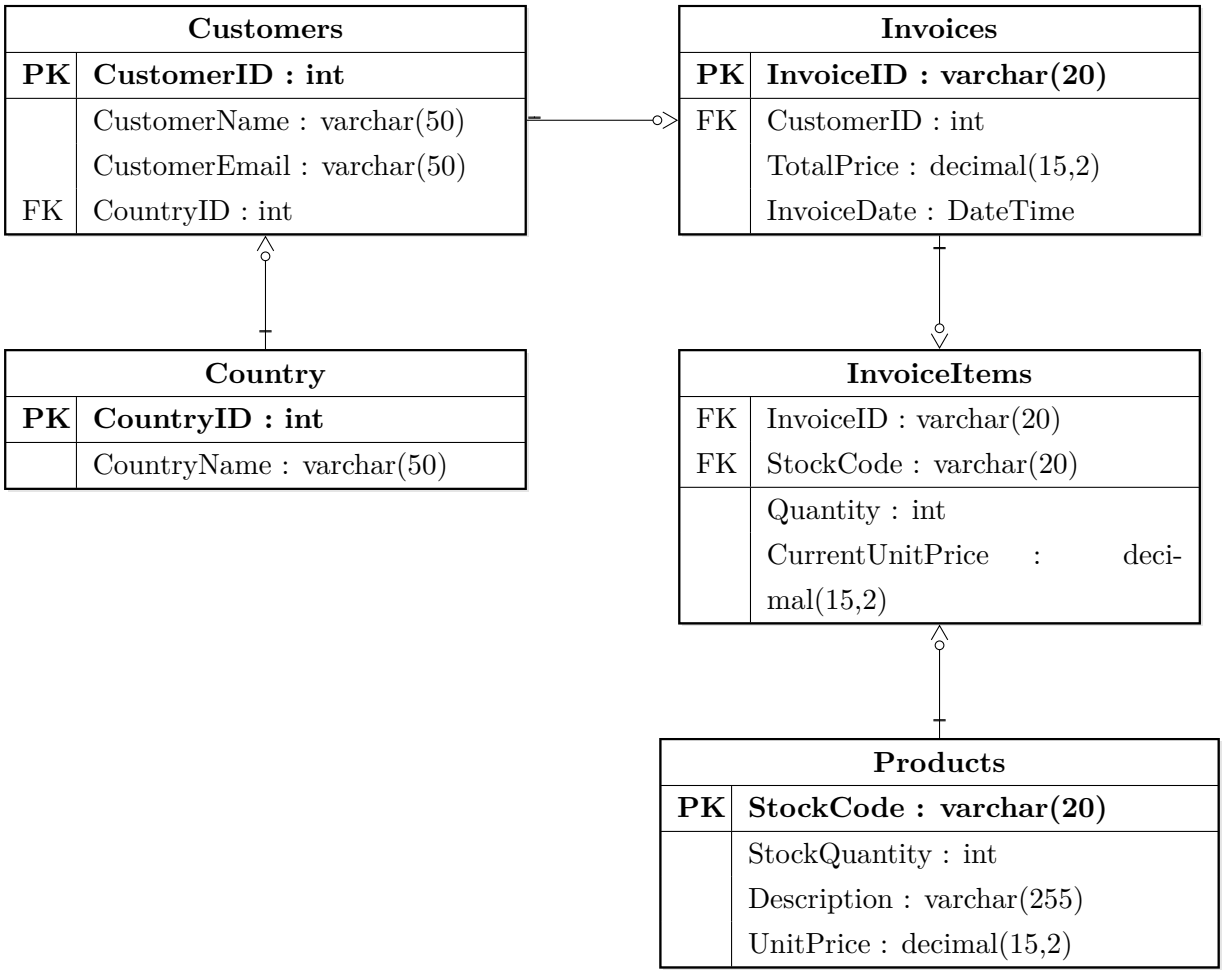
From the CSV:

- Invoice
- StockCode
- Description
- Quantity
- InvoiceDate
- Price
- CustomerID
- Country

2.2 Normalized Schema

- **Customers(CustomerID PK, CustomerName, CustomerEmail, CountryID FK)**
- **Products(StockCode PK, StockQuantity, Description, UnitPrice)**
- **Invoices(InvoiceID PK, CustomerID FK, TotalPrice, InvoiceDate)**
- **InvoiceItems(InvoiceID FK, StockCode FK, Quantity, CurrentUnitPrice)**
- **Country(CountryID PK , CountryName)**

2.3 ERD Diagram



Chapter 3

Phase 2: Database Schema (DDL)

Below is the optimized schema implementation. **Note:** Monetary values use `DECIMAL(15,2)` to ensure financial precision, replacing the imprecise `FLOAT` type.

3.1 schema.sql

```sql
-- 1. Country Table
CREATE TABLE Country (
    CountryID INT PRIMARY KEY,
    CountryName VARCHAR(50) NOT NULL UNIQUE
);

-- 2. Customers Table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50) NOT NULL,
    CustomerEmail VARCHAR(100),
    CountryID INT NOT NULL,
    CONSTRAINT FK_Customers_Country FOREIGN KEY (CountryID)
        REFERENCES Country(CountryID)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

-- 3. Products Table
CREATE TABLE Products (
    StockCode VARCHAR(20) PRIMARY KEY,
    StockQuantity INT NOT NULL DEFAULT 0 CHECK (StockQuantity >= 0),
    Description VARCHAR(255) NOT NULL,
    UnitPrice DECIMAL(15, 2) NOT NULL CHECK (UnitPrice >= 0)
);

-- 4. Invoices Table
```

```sql
CREATE TABLE Invoices (
    InvoiceID VARCHAR(20) PRIMARY KEY,
    CustomerID INT NOT NULL,
    TotalPrice DECIMAL(15, 2) DEFAULT 0.00,
    InvoiceDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT FK_Invoices_Customers FOREIGN KEY (CustomerID)
        REFERENCES Customers(CustomerID)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

-- 5. InvoiceItems Table
CREATE TABLE InvoiceItems (
    InvoiceID VARCHAR(20) NOT NULL,
    StockCode VARCHAR(20) NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity <> 0),
    UnitPrice DECIMAL(15, 2) NOT NULL CHECK (UnitPrice >= 0),
    PRIMARY KEY (InvoiceID, StockCode),
    CONSTRAINT FK_Items_Invoices FOREIGN KEY (InvoiceID)
        REFERENCES Invoices(InvoiceID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT FK_Items_Products FOREIGN KEY (StockCode)
        REFERENCES Products(StockCode)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
```

Chapter 4

Phase 3: Data Manipulation (DML)

4.1 Query 1: Insert New Product

```sql
INSERT INTO Products (StockCode, Description, UnitPrice)
VALUES ('NEW001', 'Premium Gift Box', 250000.00);
```

4.2 Query 2: Customer Order

```sql
-- Create the Header
INSERT INTO Invoices (InvoiceID, CustomerID, InvoiceDate)
VALUES ('INV90001', 13085, NOW());

-- Add Items (Triggers will handle Stock deduction and TotalPrice update)
INSERT INTO InvoiceItems (InvoiceID, StockCode, Quantity, UnitPrice)
VALUES
    ('INV90001', '85048', 2, 6.95),
    ('INV90001', '79323P', 1, 6.75);
```

4.3 Query 3: Return Item (Negative Quantity)

```sql
-- Create a new header record in the Invoices table
INSERT INTO Invoices (InvoiceID, CustomerID, InvoiceDate)
VALUES ('INV90002', 13085, NOW());

-- Insert item details into the InvoiceItems table
-- Note: A Quantity of -1 is often interpreted as a stock return
INSERT INTO InvoiceItems (InvoiceID, StockCode, Quantity, UnitPrice)
VALUES ('INV90002', '85048', -1, 6.95);
```

4.4 Query 4: Top 10 Customers by Spending

```sql
SELECT
    C.CustomerID,
    C.CustomerName,
    SUM(Ii.Quantity * Ii.UnitPrice) AS TotalSpending
FROM Customers C
JOIN Invoices I ON C.CustomerID = I.CustomerID
JOIN InvoiceItems Ii ON I.InvoiceID = Ii.InvoiceID
GROUP BY C.CustomerID, C.CustomerName
ORDER BY TotalSpending DESC
LIMIT 10;
```

4.5 Query 5: Month with Highest Revenue

```sql
SELECT
    MONTH(I.InvoiceDate) AS MonthNum,
    DATE_FORMAT(I.InvoiceDate, '%M') AS MonthName,
    SUM(Ii.Quantity * Ii.UnitPrice) AS Revenue
FROM Invoices I
JOIN InvoiceItems Ii ON I.InvoiceID = Ii.InvoiceID
WHERE YEAR(I.InvoiceDate) = 2011
GROUP BY MONTH(I.InvoiceDate), DATE_FORMAT(I.InvoiceDate, '%M')
ORDER BY Revenue DESC
LIMIT 1;
```

Chapter 5

Phase 4: Advanced SQL Features

5.1 Trigger 1: Auto-Update Inventory

Objective: Automatically deduct stock from the Products table when an item is added to an invoice.

```
DELIMITER $$

CREATE TRIGGER trg_UpdateInventory_AfterInsert
AFTER INSERT ON InvoiceItems
FOR EACH ROW
BEGIN
    UPDATE Products
    SET StockQuantity = StockQuantity - NEW.Quantity
    WHERE StockCode = NEW.StockCode;
END$$

DELIMITER ;
```

5.2 Trigger 2: Auto-Update Invoice Total

Objective: Recalculate the Invoice TotalPrice whenever an item is added, removed, or modified.

```
DELIMITER $$

-- Trigger for Insert
CREATE TRIGGER trg_UpdateInvoiceTotal_Insert
AFTER INSERT ON InvoiceItems
FOR EACH ROW
BEGIN
    UPDATE Invoices
    SET TotalPrice = (
```

```
        SELECT COALESCE(SUM(Quantity * UnitPrice), 0)
        FROM InvoiceItems
        WHERE InvoiceID = NEW.InvoiceID
    )
    WHERE InvoiceID = NEW.InvoiceID;
END$$

-- Trigger for Delete (Crucial for data integrity)
CREATE TRIGGER trg_UpdateInvoiceTotal_Delete
AFTER DELETE ON InvoiceItems
FOR EACH ROW
BEGIN
    UPDATE Invoices
    SET TotalPrice = (
        SELECT COALESCE(SUM(Quantity * UnitPrice), 0)
        FROM InvoiceItems
        WHERE InvoiceID = OLD.InvoiceID
    )
    WHERE InvoiceID = OLD.InvoiceID;
END$$

DELIMITER ;
```

5.3 Stored Procedure: Customer History Report

```
DELIMITER $$

CREATE PROCEDURE GetCustomerInvoiceHistory(IN cid INT)
BEGIN
    SELECT
        I.InvoiceID,
        I.InvoiceDate,
        COUNT(Ii.StockCode) as TotalItems,
        I.TotalPrice
    FROM Invoices I
    LEFT JOIN InvoiceItems Ii ON I.InvoiceID = Ii.InvoiceID
    WHERE I.CustomerID = cid
    GROUP BY I.InvoiceID, I.InvoiceDate, I.TotalPrice
    ORDER BY I.InvoiceDate DESC;
END$$

DELIMITER ;
```