Retail Store Database

By: Nicholas Sutton, Mae Thomas, and Luke Bailey
Professor: Jalal Omer
Course: CS 4347
8/4/2024

Table of Contents

Introduction

Problem Statement

Retail stores deal with huge quantities of information concerning products, sales, clients, stock and labor force. Proper information handling aims at perfecting operations, enhancing client service delivery and making informed business choices. Nevertheless, challenges encountered when using excel for these purposes include:

1. **Volume of Data:** Excel is not able to handle big datasets; this results in slow processing speeds and possible data corruption. Increasing the size of the datasets causes excel to slow down more, hence being prone to mistakes.
2. **Data Validity:** There are no strong mechanisms for ensuring that the inputted information is accurate or consistent throughout various cells in excel. This may lead to wrong analysis reports.
3. **Collaboration:** Many people using one file at a time can create versioning problems where individuals may override each other's changes unknowingly, leading to loss of some or all data contained within such files.
4. **Scalability:** The amount of data becomes unmanageable within excel as the business grows larger with time. When datasets grow large and relationships between them become more complicated, excel cannot cope.
5. **Security:** Excel has very few built-in security features; thus sensitive information can easily be seen by unauthorized persons. It is possible for anybody who does not have permission to open or modify such a file.

In order to deal with these problems in retail data management you should have a well-organized scalable environment made by a database system.

Solution

1. **Efficiency and Performance:** When it comes down to efficiency and performance; databases are designed to handle large amounts of data efficiently. They are capable of supporting complicated searches which can be indexed for quick access as well as modification thus greatly boosting their performance over spreadsheets.
2. **Data Integrity:** In terms of data integrity, it is maintained through various constraints imposed by databases such as primary keys, foreign keys and unique constraints among others so that there would be few mistakes or none at all while still keeping everything uniform.
3. **Collaboration:** More than one person can access the database at the same time without any conflicts arising between users who might have edited some information that others were not aware about.
4. **Scalability:** Databases can easily expand to incorporate larger data volumes and more complex designs, making them suitable for growing businesses. This is done by efficient handling of big datasets and challenging queries.

5. **Security:** Databases come with enhanced security features like user authentication, access controls and data encryption which safeguard sensitive information from being accessed without authorization or breached.

Target Users

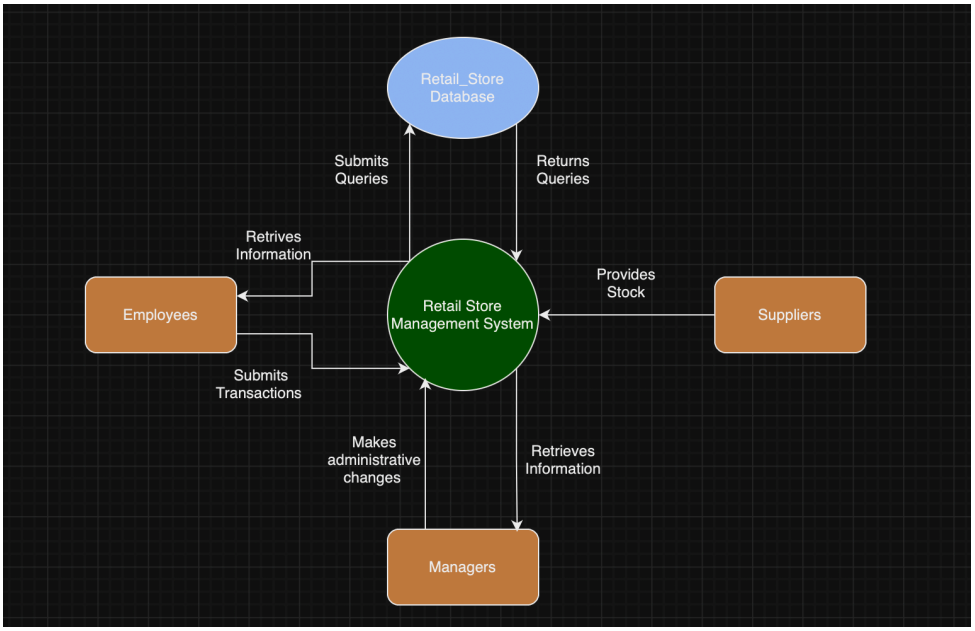
- **Store Managers:** Access to sales reports and inventory levels is needed for proper stocking decisions. Additionally, customer data is necessary for promotion and staffing choices by managers.
- **Sales Staff:** Real-time inventory information should be at their fingertips in order to help customers and process sales quickly.

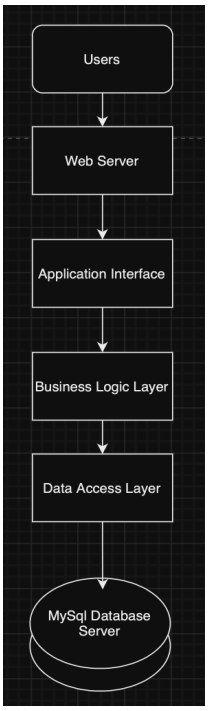
Organization of this Report

This report is designed to highlight each step in the development process of our project. The report flows in chronological order, walking the reader through much of the content that was turned in as deliverables, as well as some extra analysis and reflection. The sections of this report from system requirements to functional dependencies and database normalization all break down the preliminary planning process, while sections The Database System and User Application Interface focus on the implementation of our project.

System Requirements

Context Diagrams





Interface Requirements

- Visually appealing UI
- Data Validation
- User Friendly
- Essential CRUD operations
- Page navigation

Functional & Non-Functional Requirements

Functional Requirements

- Employee Management
- Product Management
- Supplier Management
- Transaction Management
- Inventory Management

Non-Functional Requirements

- Efficiency
- Reliability
- Adaptability
- Versatility
- Portability

Conceptual Design of the Database

ER Diagram for the Relational Database

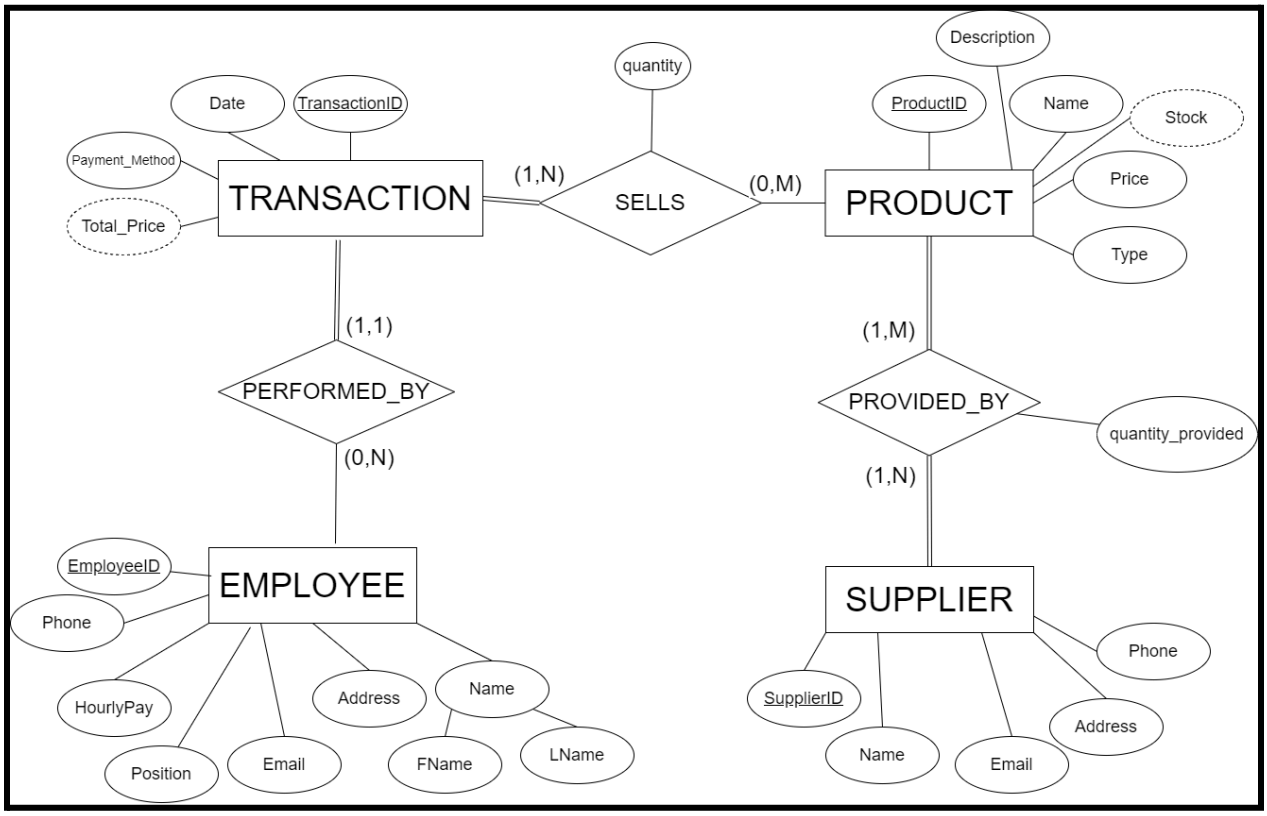


Figure 2.1: Final entity relationship diagram.

Data Dictionary

Custom Domains

Name	**Type**
PHONE_TYPE	VARCHAR(15)
SSN_TYPE	CHAR(9)
EMAIL_TYPE	VARCHAR(50)
ADDRESS_TYPE	VARCHAR(120)
NAME_TYPE	VARCHAR(40)
PAYMENT_TYPE	VARCHAR(40)

Products

Attribute	Key?	Description	Purpose	Data Type	Default	Actions
ProductID	PK	Surrogate key	To act as the PK	INT	NOT NULL	AUTO INCREMENT
SupplierID	FK	The ID of the supplier that supplies this product	To know who supplied the product	INT	NOT NULL	ON DELETE: SET NULL ON UPDATE: CASCADE
Name	no	Name of the product	To know the name of the product to not have to rely solely on ProductID	NAME_TYPE	“”	N/A
Price	no	Cost of the product in USD	To know how much a specific good costs	FLOAT	0.0	N/A
Stock	no	Amount of product in inventory	To know if it is possible to sell any more of a product	INT	0	N/A
Description	no	Simple english description of what the product is	To get information about what a product is	VARCHAR(120)	“”	N/A

Transactions

Attribute	Key ?	Description	Purpose	Data Type	Default	Actions
TransactionID	PK	Surrogate key	To act as the PK	INT	NOT NULL	AUTO INCREMENT
Date	no	Month/Day/Year and time that the transaction was processed on	To know when the transaction occurred	DATETIME	should be current date/ time when created	N/A
Payment_Method	no	The type of payment (cash, card, etc.) that was used to pay for the transaction	To know what kind of payment was used for the transaction	PAYMENT_TYPE	Cash	N/A
Total_Price	no	The total cost of the entire transaction (usually the quantity * the product's price)	To keep track of the cost of a transaction, as it may be different than simply the numerical product in the event of a sale, etc.	FLOAT	0.0	N/A
EmployeeID	FK	The ID of the employee processing the transaction	To know which employee the transaction was processed by	INT	NULL	ON DELETE: SET NULL ON UPDATE: CASCADE

Employees

Attribute	Key?	Description	Purpose	Data Type	Default	Actions
EmployeeID	PK	Surrogate key	To act as the PK	INT	NOT NULL	AUTO INCREMENT
FName	no	First name of the employee	To know name of employee	NAME_TYPE	“”	N/A
LName	no	Last name of the employee	To know name of employee	NAME_TYPE	“”	N/A
Address	no	The employee’s home address	To know where the employee lives for administrative uses	ADDRESS_TYPE	“”	N/A
Phone	no	The employee’s phone number	To contact the employee	PHONE_TYPE	“”	N/A
Email	no	The employee’s email address	To contact the employee	EMAIL_TYPE	“”	N/A
Position	no	The employee’s position within the company (cashier, inventory, etc)	To know what the employee does at the company	VARCHAR(15)	“”	N/A
Hourly_Pay	no	The amount of USD the employee earns per hour	To know how much the employee earns	FLOAT	7.25	N/A

note: hourly pay is national minimum wage

Suppliers

Attribute	Key?	Description	Purpose	Data Type	Default	Actions
SupplierID	PK	Surrogate key	To act as the PK	INT	0	AUTO INCREMENT
Name	no	The name of the supplier	To know the name	NAME_TYPE	“”	N/A
Address	no	The address the supplier is located at	To know where the supplier is based from	ADDRESS_TYPE	“”	N/A
Phone	no	The phone number of the supplier	To contact the supplier for any information, etc. about products	PHONE_TYPE	“”	N/A
Email	no	The email address of the supplier	To contact the supplier for any information, etc. about products	EMAIL_TYPE	“”	N/A

Sells

Attribute	Key?	Description	Purpose	Data Type	Default	Actions
TransactionID	FK	Candidate Key	To show which transaction the tuple is apart of	INT	“”	ON DELETE: CASCADE
ProductID	FK	Candidate Key	To show which product the tuple is referring to	INT	“”	ON DELETE: CASCADE
Quantity	no	Quantity of the product sold in the transaction	To keep track of stock, and calculate price of transaction	INT	“”	N/A

Provided_By

Attribute	Key?	Description	Purpose	Data Type	Default	Actions
SupplierID	FK	Candidate Key	To show which supplier provided the product	INT	“”	ON DELETE: CASCADE
ProductID	FK	Candidate Key	To show which product is being provided	INT	“”	ON DELETE: CASCADE
Quantity_Provided	no	Quantity of the product sold in the transaction	To update stock	INT	“”	N/A

Business Rules

1. **Primary Keys:** Primary keys for tables Products, Employees, Suppliers, and Transactions all automatically increment. Sells and Provided_By relations use composite keys to provide unique identification.
2. **Total Price Calculation:** The Total_Price attribute of products is a derived value from adding the price of each product times the quantity.
3. **Stock Management:** Product stock is subtracted from when a transaction involving that product is performed and product stock is added to when a supplier provides more of the product.

Logical Database Schema

Database Schema

The schema below was adapted from our ER diagram (fig 2.1). We went through many variations of this schema while deciding on the exact components that we wanted to include in our database. In addition to this, the conversion into 3rd Normal Form required the addition of two new relational tables: Sells and Provided_By.

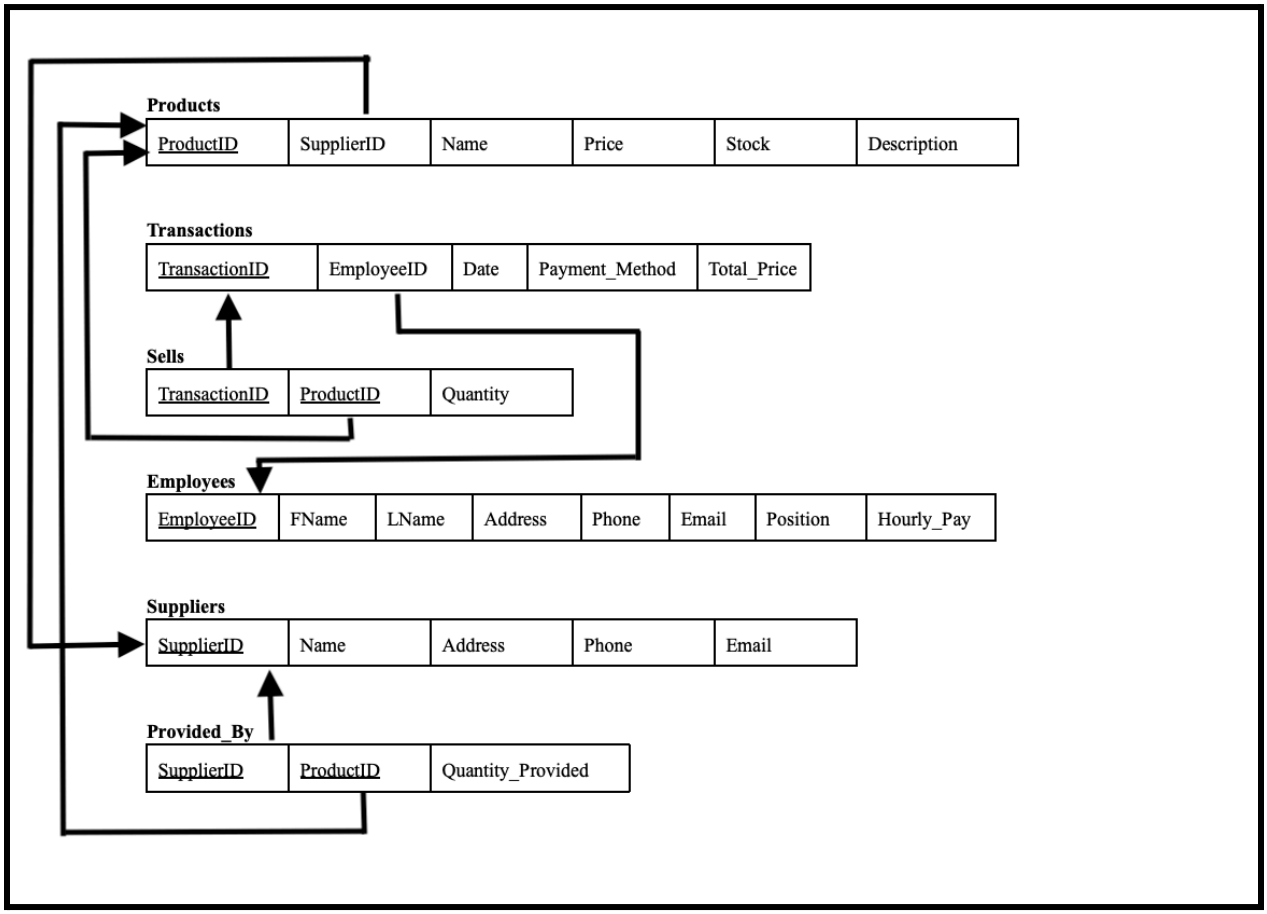


Figure 3.1: Final relational schema

Expected Operations

Products: Create, Read, Update, Delete

Transactions: Create, Read, Delete

Suppliers: Create, Read, Update, Delete

Employees: Create, Read, Update, Delete

Restock (Provided_By): Create

Sells: Create

Expected Data Volumes

Our database is ideal for relatively small retail store operations. Because of this, most tables will have a fairly low data volume. Below is the expected data volumes for each table from lowest volume to highest volume.

Employees: Likely < 40 entries at a time.

Suppliers: Likely < 75 entries at a time.

Products: Likely < 100 entries at a time.

Restock (Provided_By): Will never stop increasing, however at a fairly slow rate.

Transactions: One of the highest volume tables as it will never stop increasing.

Sells: The highest volume table as it will require an entry for every product sold in a transaction.

Create.Sql Script

In this phase of the project we also began writing our create.sql file which is meant to create the tables as well as their attributes. In addition to creating the necessary tables, this sql script also includes triggers to automatically update the stock of a product each time a transaction involving that product is performed or the supplier provides more of it. Below you can find our sql code.

```
CREATE DATABASE IF NOT EXISTS Retail_Store;
USE Retail_Store;

CREATE TABLE Employees (
  EmployeeID INT PRIMARY KEY,
  FName VARCHAR(40),
  LName VARCHAR(40),
  Address VARCHAR(120),
  Phone VARCHAR(15),
  Email VARCHAR(50),
  Position VARCHAR(15),
  Hourly_Pay FLOAT DEFAULT 7.25
);


CREATE TABLE Suppliers (
  SupplierID INT PRIMARY KEY,
  Name VARCHAR(40),
  Address VARCHAR(120),
  Phone VARCHAR(15),
  Email VARCHAR(50)
);


CREATE TABLE Products (
  ProductID INT PRIMARY KEY,
  SupplierID INT,
  Name VARCHAR(40),
  Price FLOAT DEFAULT 0.0,
  Stock INT DEFAULT 0,
  Description VARCHAR(120),
  FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID) ON DELETE SET NULL ON UPDATE CASCADE
);


CREATE TABLE Transactions (
  TransactionID INT PRIMARY KEY,
  EmployeeID INT,
  Date DATETIME,
  Payment_Method VARCHAR(40) DEFAULT 'Cash',
  Total_Price FLOAT DEFAULT 0.0,
  FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID) ON DELETE SET NULL ON UPDATE CASCADE
);


CREATE TABLE Sells (
  TransactionID INT,
  ProductID INT,
  Quantity INT,
  PRIMARY KEY (TransactionID, ProductID),
  FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID),
  FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);


CREATE TABLE Provided_By (
  SupplierID INT,
  ProductID INT,
  Quantity_Provided INT,
  PRIMARY KEY (SupplierID, ProductID),
  FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID),
  FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);


DELIMITER //

CREATE TRIGGER UpdateStockSELL
AFTER INSERT ON Sells
FOR EACH ROW
BEGIN
   UPDATE Products
   SET Stock = Stock - NEW.Quantity
   WHERE ProductID = NEW.ProductID;
END//

CREATE TRIGGER UpdateStockIMPORT
AFTER INSERT ON Provided_By
FOR EACH ROW
BEGIN
   UPDATE Products
   SET Stock = Stock + NEW.Quantity_Provided
   WHERE ProductID = NEW.ProductID;
END//

DELIMITER ;
```

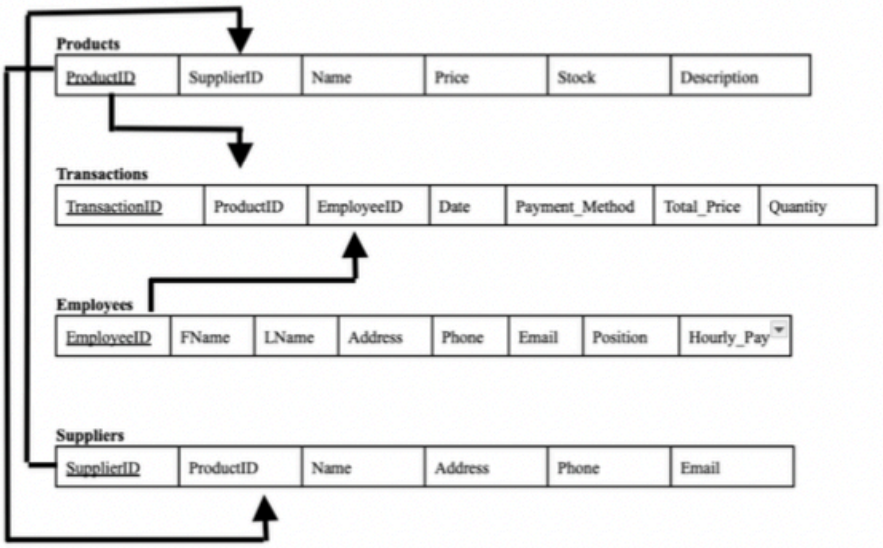
Figure 3.2: Create.sql Script

Functional Dependencies and Database Normalization

Normalization Process

Our original schema included 4 tables: Products, Transactions, Employees, and Suppliers. Figure 4.1 contains our pre-normalized schema as well as the corresponding functional dependencies.

I. Original Schema



II. Original Functional Dependencies

Products:
ProductID → SupplierID, Name, Type, Price, Stock, Description

Transactions:
TransactionID → ProductID, EmployeeID, Date, Payment_Method, Total_Price, Quantity

Employees:
EmployeeID → FName, LName, Address, Phone, Email, Position, Hourly_Pay

Suppliers:
SupplierID → ProductID, Name, Address, Phone, Email

Figure 4.1: Original schema and functional dependencies

1. Products
    - The products table satisfies all the requirements to be in 1NF
    - There are no partial dependencies, therefore the products table is in 2NF.
    - There are no transitive dependencies, therefore the products table is in 3NF.
2. Transactions
    - The transactions table is not in 1NF because ProductID can be a multivalued attribute. To remedy this, a separate relation, “sells”, will be created using TransactionID and ProductID as the keys
    - After making the previous change, there are no partial dependencies in either table therefore they are in 2NF.
    - There are no transitive dependencies, therefore the transactions and sales tables are in 3NF.
3. Employees
    - The employees table satisfies all the requirements to be in 1NF.
    - There are no partial dependencies, therefore the products table is in 2NF.
    - Because SSN is not being used as a key, and solely a unique attribute, there are no transitive dependencies so the employees table is in 3NF.
4. Suppliers
    - The suppliers table is not in 1NF because ProductID can be multivalued. To remedy this, a separate relation, “provided_by”, will be created using SupplierID and ProductID as the keys.
    - After making the previous change, there are no partial dependencies in either table therefore they are in 2NF.
    - There are no transitive dependencies, therefore the transactions and provided_by tables are in 3NF.

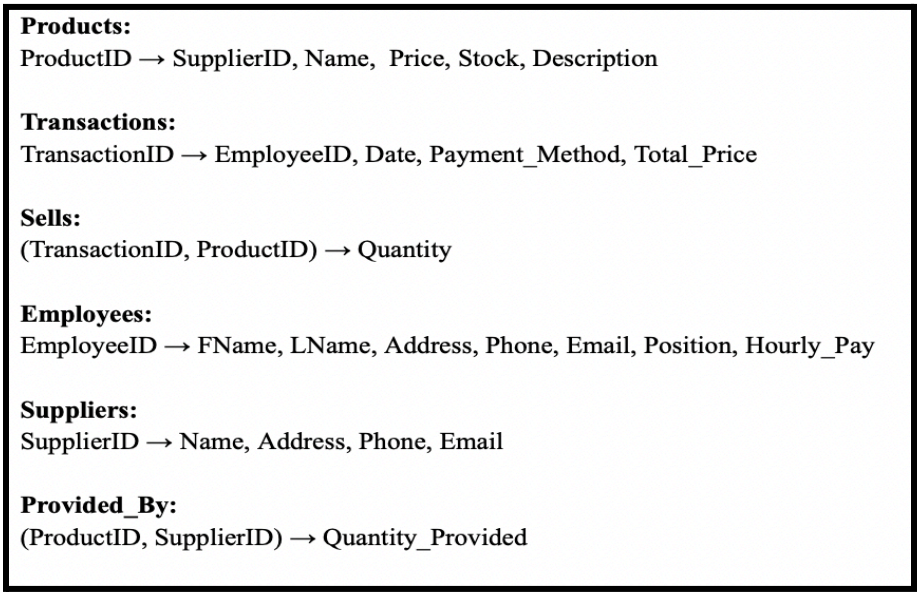


Figure 4.2: Final functional dependencies

The Database System

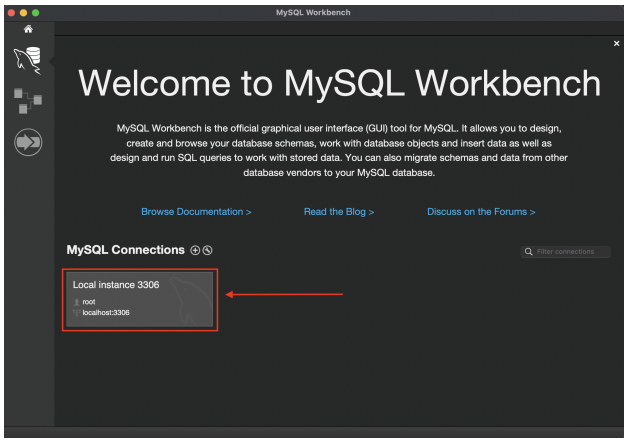
Installing and Invoking our System

Installing our system is fairly simple as long as you use the right software. It's worth noting that many different softwares can be used, however this section will walk you through the way that our team implemented this project.

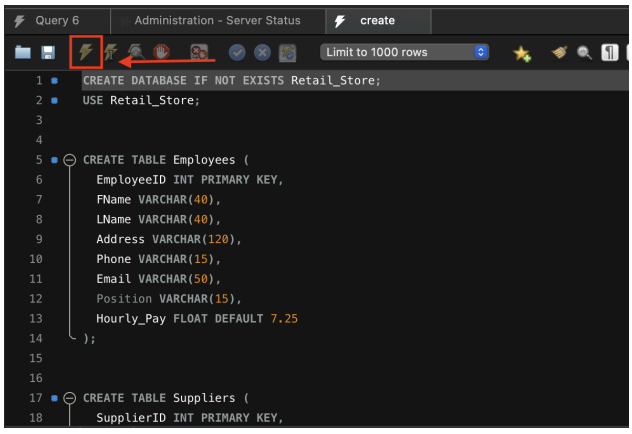
Suggested Software: MySql, MySql Workbench, VSCode, NextJS, NodeJS

Step 1: Setting up the database

Upon installation of MySql and MySql Workbench, open the workbench application and either select the Local Instance provided for free by the MySql Community version, or connect to any other desired instance. Our team used the local instance as it sufficed for our needs, however a business implementing our project might want to consider a cloud-based server.



Once you're connected, run our file 'create.sql'. To do this navigate to File < Open SQL Script < select the file < click the lightning bolt to run the script. You can now right click on the left hand navigation bar and select refresh schema and you should be able to see the tables and all of their constraints and attributes.



Step 2: Setting up the user interface

To set up this user interface application on your personal system, first you need to install NodeJS to be able to run the application. This can be done via a simple install on the NodeJS website. Once this has been installed open the project folder in an IDE such as VSCode. Go to the terminal and run the following command once you are in the product directory: “npm run dev”.

```
\database project> npm run dev
```

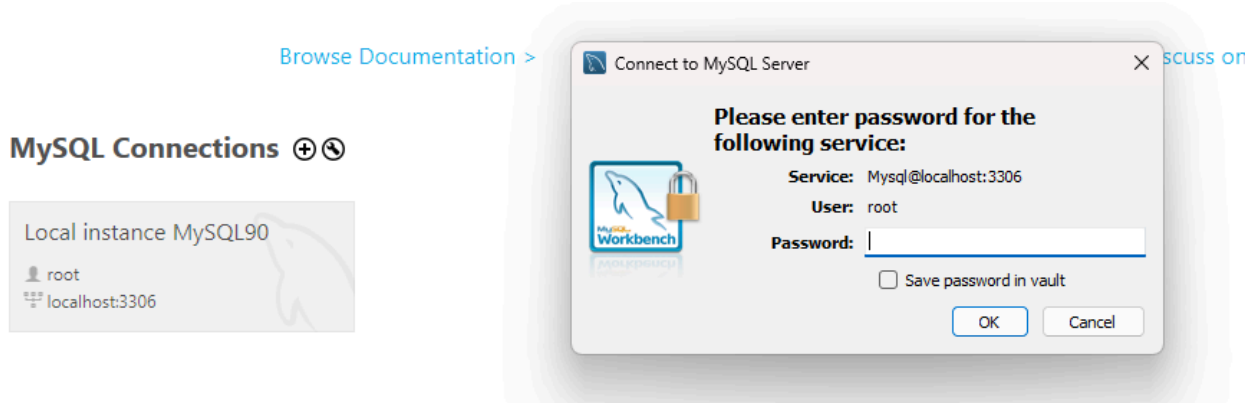
This will start the server on localhost port 3000 where you can view the user interface by visiting http://localhost:3000.

```
> database-project@0.1.0 dev
> next dev

   ▲ Next.js 14.1.0
   - Local:        http://localhost:3000

 ✓ Ready in 1958ms
```

If you are missing any dependencies type “npm install” to install the remainder.

Step 3: Putting it all together

To launch the database in sync with the user interface to the above steps to launch the NodeJS server. Then go onto MySql Workbench to start the database server.



Type in the password for the server and then the database should automatically be connected to the user interface without any additional setup.

User Application Interface

Interface Development Process

The interface was designed using TailwindCSS, Mantine, and NextJS for the component aspect. We decided upon a simple user interface that consisted mainly of a homepage which would allow users to quickly direct themselves to the important areas of the application. These areas were the product page, employee page, transaction page, and supplier page. To compartmentalize the code all page-related components were sectioned off into the /components folder. Within this folder there are sub-folder each titled with the page the components are being used. Styling of these components was done with a combination of the Mantine component library and TailwindCSS for more fine-tuned styling purposes. For the API related functions, these are stored inside the /api folder. These functions connect the components to the database.

Interface Navigation and Utilization

Our interface is very user friendly so it's fairly simple to navigate and use. Upon starting the application, the user will see a navigation page with buttons leading to pages for: Transactions, Products, Suppliers and Employees. Selecting one of these options will lead the user to the designated page where all previous database entries will already be displayed. In addition to this, each page has its own create, update, and delete operations which can be performed by selecting the option and filling in any necessary fields. The user can switch between these pages simply by going back to the main navigation page.

Final Thoughts

Conclusion

This report highlights the entire development and implementation process for our Database Project. Our task was to create a fully functioning, efficient, and normalized database as well as an interface that will be interacted with by our target users. Throughout the process of completing this project, our team was able to get a feel for what working on a larger scale project in a group would be like. As a result, we prioritized delegation, collaboration, and open communication to ensure our work remained cohesive and consistent. In addition to this, tasks like drafting SQL queries and practicing normalization also gave us valuable insight and experience parallel to the material that we have learned in CS 4347 this semester.

Future Work

This project is meant to be a general framework for any retail store. The simplicity allows for anyone implementing it to put their own spin on it, and make any adjustments that better fit their business. Because of this, future work on this project can be almost anything. For future work that we could do on this project, there are potentially limitless possibilities to implement into the system. Some examples include: adding a payroll system, user login, and adding more query elements to the user interface.

References

Elmasri, Ramez, and Sham Navathe. *Fundamentals of Database Systems*. Pearson, 2016.

Appendix: Zip File Contents

I. /doc
   - DBProjectPresentation.pdf
   - Phase1Report.pdf
   - Phase2Report.pdf
   - taskA.pdf
   - taskb.pdf
   - Taskc.pdf
   - FinalReport.pdf

II. /project
   - create.sql
   - load.sql
   - Products.csv
   - Provided_By.csv
   - Suppliers.csv
   - Employees.csv
   - /front end

III. README file