DATABASE MANAGEMENT SYSTEM

PROJECT REPORT

Group Members:

- **Syed Hassan Raahim, 01-134211-102**
- **Usman Saeed, 01-134211-071**
- **Usman Waheed, 01-134211-098**

Department of Computer Science

BAHRIA UNIVERSITY, ISLAMABAD

Title: DRIPPY TOES DATABASE

About The Business:

Drippy Toes is a shoe store that sells pre-loved/used shoes on Instagram.

Referral Code scheme:

The business provides its prospects with a referral code after the prospect becomes a customer. The customers can earn from their codes by sharing it with other people.

When another prospect or customer uses another customer's code, they get a discount on their order and the customer (whose code was used) gets a certain amount of cash.

The code has 12 uses. The uses of a customer's code get reset after they make another purchase.

How shoes are bought:

The shoes are bought from various suppliers. Since the shoes are pre-loved/previously used, they have service expenses such as cleaning, repair, etc., which are done by service providers.

How orders are placed:

The orders are placed on Instagram. A prospect sends a message to the Instagram Profile of the store and an admin replies to assist them. When an order is placed, the admin inserts the name of the customer in the Excel sheet next to the names of the shoes the prospect bought. This marks the shoes in the Excel sheet as sold. Along with customer name being entered, the customer code is also generated and place in the sheet.

Inventory Management:

After the shoes are bought from the supplier, the admin enters them into the Excel sheet with the details and cost. Details include information such as date, name, size etc. The post date, which is the date when a shoe is posted on the Instagram platform is also written. If a shoe has a customer name in its corresponding column, then this means the shoe have is sold.

Marketing:

The brands markets itself and its shoes through various marketing campaigns on various mediums.

How business reports are generated:

The admin generates various reports by applying Excel function to the data in the Excel sheet.

Problem:

Business Need:

They are currently using Microsoft Excel to manage their business and are in need of a DBMS system instead.

The problems being faced :

Redundancy:

Since all the partners require an updated record, whenever the record gets updated it has to be duplicated and sent to the partners.

Availability:

Since the person in charge of the record only holds the copy, the partners have to wait for him to send the updated version instead of instantly seeing the changes he has made.

Isolation of data:

A limitation of excel is that data cannot be isolated into certain subsets and views cannot be created by arranging data.

Fixed Queries:

The biggest problem the sales team is facing is that there is no way to calculate how much a customer has bought from the store (Customer equity) with a single query. In excel the would have to search the customer's name at different instances where they bought the product and then calculate manually.

How DBMS fixed this:

Redundancy:

Well, the database can be hosted on a central server where any change made would be instantly updated and available for everyone to view.

Availability:

Since data would be on a server a single link would be generated from which anyone can use at any time hence fixing this problem.

Isolation of data:

We can easily subset data and create views in DBMS.

Fixed Queries:

Easily be fixed by creating a relation in the database for the customer which has the set attribute (Customer Equity).

Future Work:

The database created will be further used to create a front-end management app which the admin will use instead of Excel.

The admin would not need to write Excel formulas as the database would consist of views. The database information would be accessible to all other partners since it would be hosted on a server. The copy of the Excel sheet would not have to be shared every time there is an update made to it.

The users of the database are:
Admin (Can Create, Read, Update, Delete)
Partners (Read Only Specific Information)

Views:
Monthly Report
Inventory
Weekly Report

Data Requirements: (How the data was gathered)

Step 1:

Data requirements were gathers using the excel sheet of the business and the by studying about the business itself and its underlying workings.

Step 2:

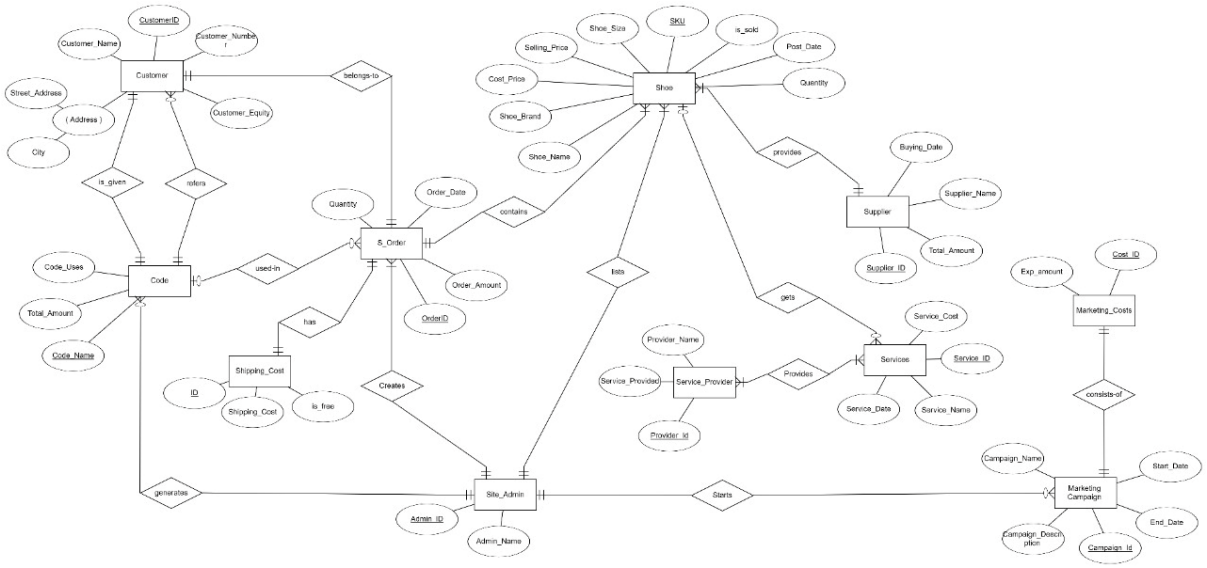
Using the knowledge we had obtained in the previous step, we formed entities and gave them meaningful attributes and relationships.

Tables:

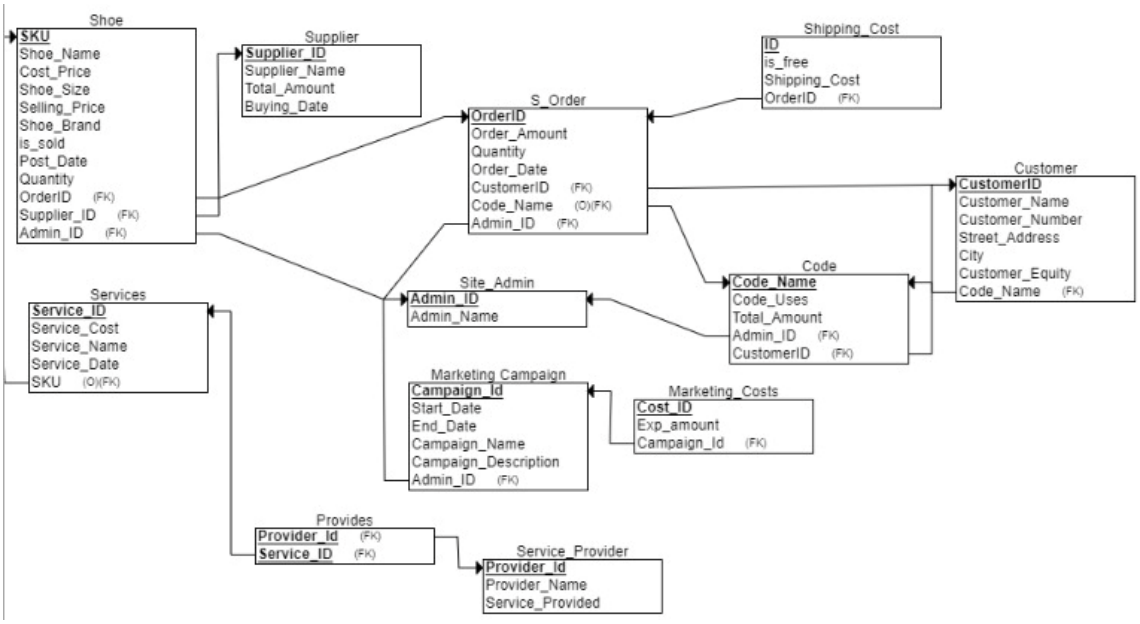
- **Customer**
- **S_Order**
- **Shoe**
- **Code**
- **Services**
- **Provides**
- **Service_Provider**
- **Site_Admin**
- **Marketing_Campaign**
- **Marketing_Costs**
- **Shipping_Cost**
- **Supplier**

Then based on the ERD, we created a relational mapping schema. Which is created by adding foreign keys and mapping them with primary keys.

Entity-Relationship Diagram:



Relational Schema:



Tuples:

Shoe (SKU, Shoe_Name,Cost_Price, Shoe_Size, Selling_Price, Quantity, Shoe_Brand, is_sold, Post_Date, OrderID, Supplier_ID, Admin_ID)

S_Order (OrderId, Order_Amount, Quantity, Order_Date, CustomerID, Code_Name, Admin_ID)

Shipping_Cost (ID, is_free, Shipping_Cost, OrderID)

Code (Code_Name, Code_Uses, Total_Amount, Admin_ID, CustomerID)

Customer (CustomerID, Customer_Name, Customer_Address, Customer_Number, Street Address, City, Customer_Equity, Code_Name)

Admin(Admin_ID, Name)

Marketing_Campaign (Campaign_ID, Start_Date, End_Date, Campaign_Name, Campaign_Description, Admin_ID)

Marketing_Costs (Cost_ID, Exp_amount, Campaign_ID)

Supplier (Supplier_ID, Supplier_Name, Total_Amount, Buying_Date)

Services (Service_ID, Service_Cost, Service_Name, SKU, Service_Date)

Provides (Provider_ID, Service_ID)

Service_Provider(Provider_ID, Provider_Name, Service_Provided)

Marketing_Cost_Campaign (Campaign_ID, Cost_ID)

Shoe_Order (SKU, OrderID)

Shoe_Supplier (SKU, Supplier_ID)

Code_Customer (Code_Name, CustomerID)

Normalization:

First Normal Form (1NF) :

Condition:

All the fields in the table should contain only scalar values.

Steps to convert a table into 1NF;

Step 1: Place all items that appear in the repeating group in a new table.

Step 2: Designate a primary key for each new table produced.

Step 3: Duplicate in the new table the primary key of the table from which the repeating group was extracted.

Explanation:

To convert the database in the 1NF we checked for any items that are repeating in any table. When we found some repeating items we created a new table for it. After which we assign it a new primary key and the primary key of the table from which it was extracted.

Second Normal Form (2NF) :

Condition:

• The Table is in first normal form.

• All nonkey attributes in the table must be functionally dependent on the entire primary key.

Steps to convert into 2NF:

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.

2. If other data items are functionally dependent on the same part of the key, place them in the new table.

3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table.

Explanation:

The tables in our database are in 2NF as all the fields of the tables contain scalar values and all the nonkey attributes in tables are functionally dependent on the primary key.

Third Normal Form (3NF) :

Condition:

• The table should be in second normal form.

• No attribute is transitively dependent on the primary key.

Steps to convert into 3NF:

1. Move all items involved in transitive dependencies to a new entity.
2. Identify a primary key for the new entity.
3. Place the primary key for the new entity as a foreign key on the original entity.

Explanation:

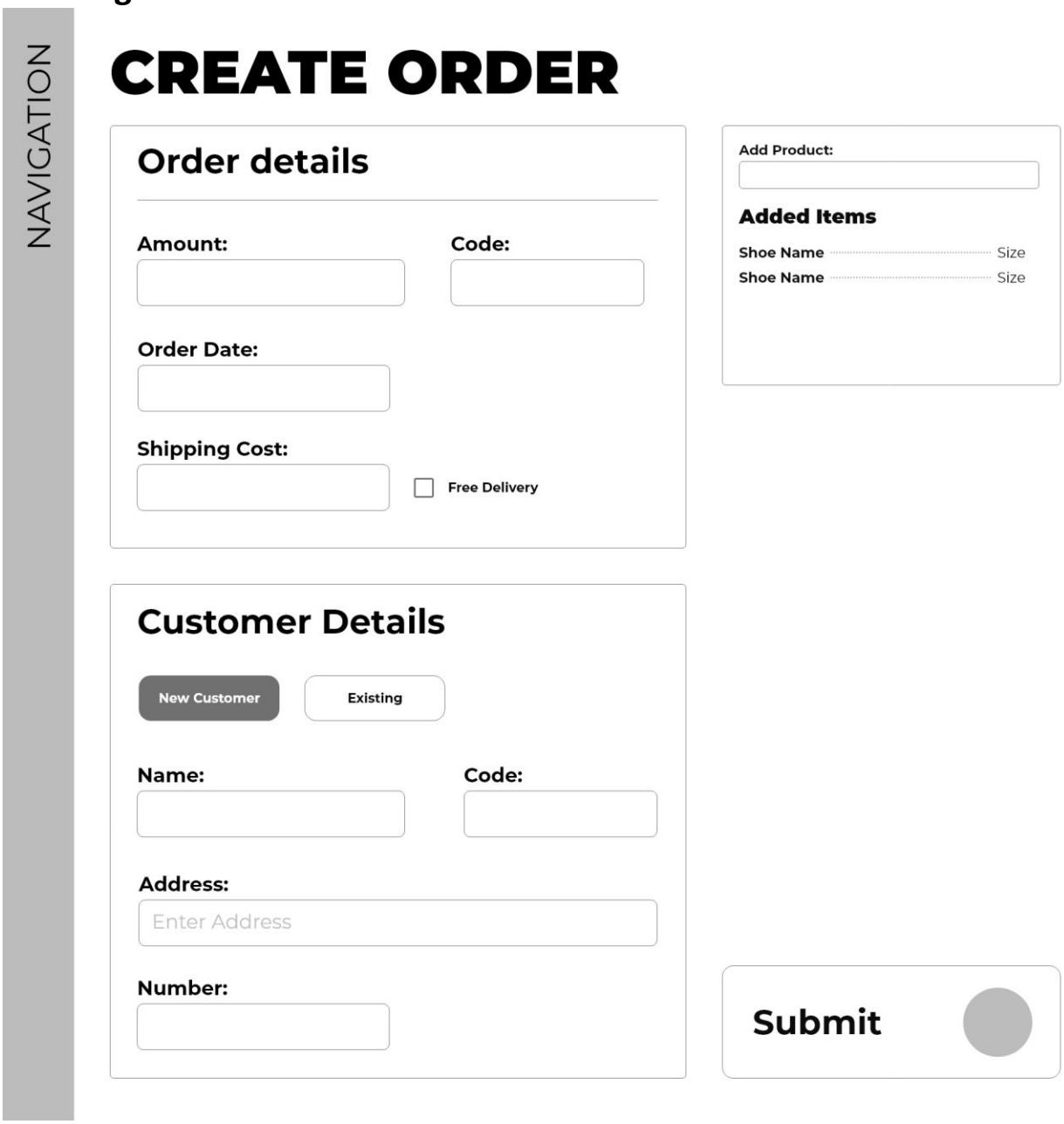
The Items in ***Shoe and Marketing Campaign*** were involved in transitive dependencies so we have created a new entity (tables) for them and move them in it. The new tables created are ;

- Marketing_Campaign_Admin
- Marketing_Cost_Campaign
- Shoe_Order
- Shoe_Supplier
- Shoe_Admin
- S_Order_Code
- S_Order_Admin
- Code_Admin
- Code_Customer

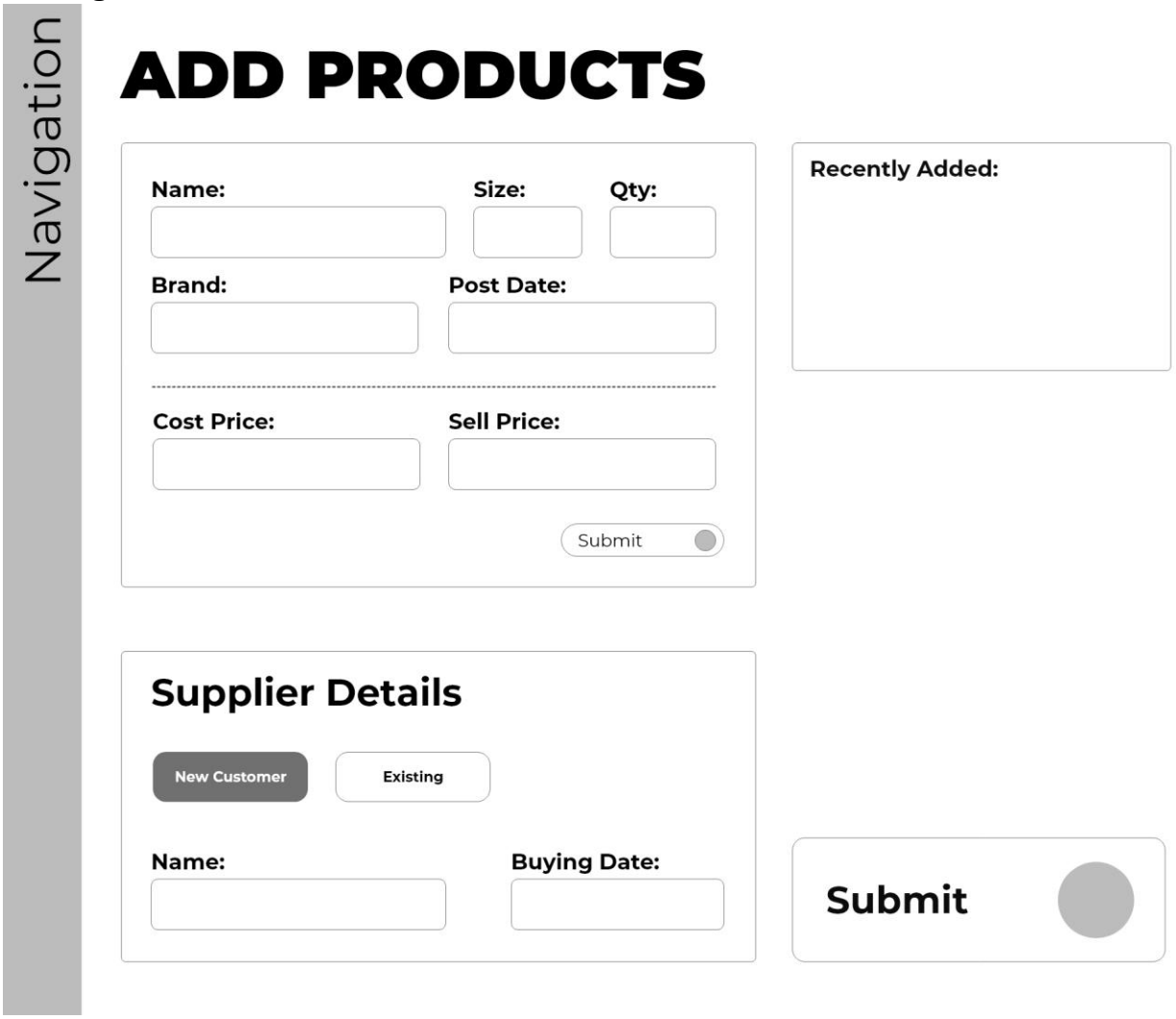
Wireframes:

Consists of the design of the forms that will be used by the user.

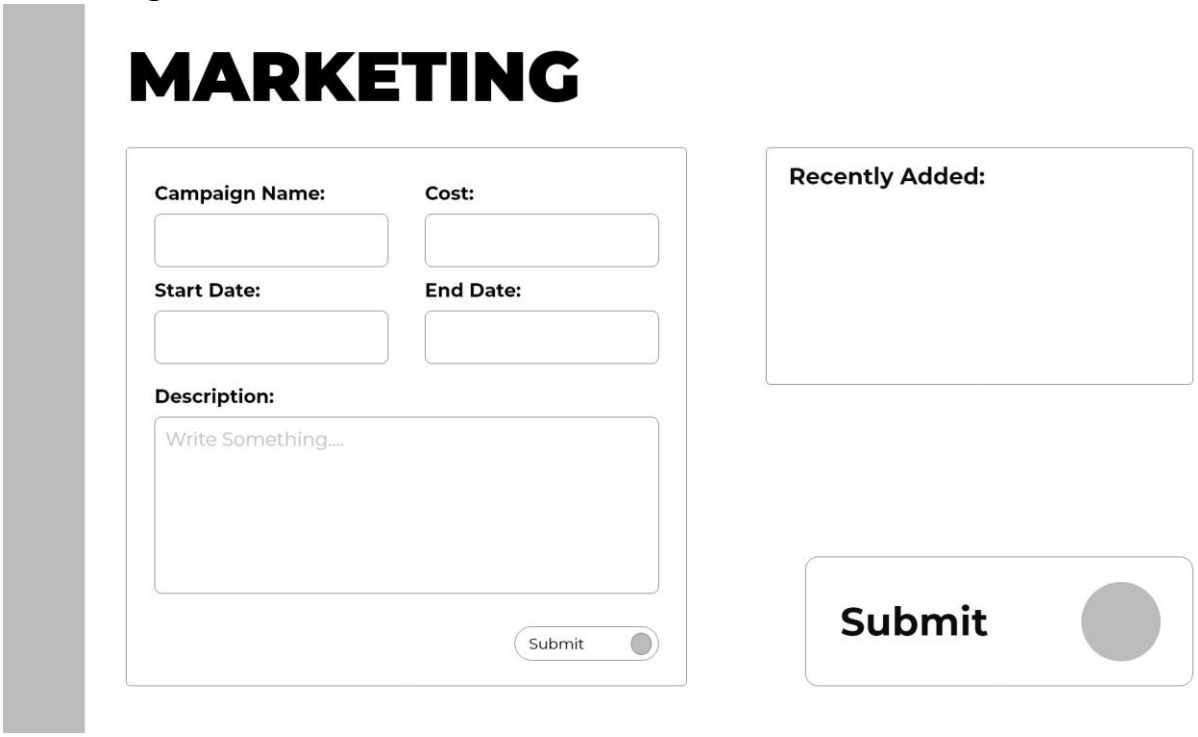
Creating an Order:



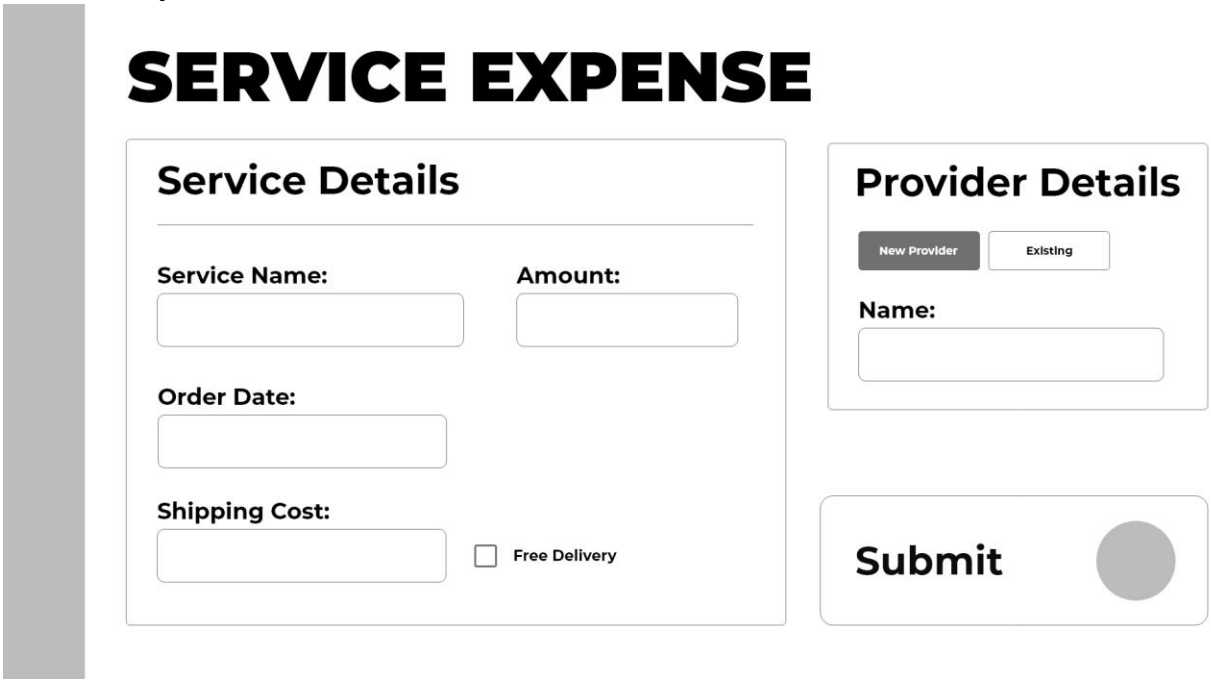
Adding Products:



Marketing Information:



Service Expenses:



Script:

```
SUPPLIER TABLE
CREATE TABLE Supplier
(
 Supplier_Name VARCHAR(255) NOT NULL,
 Supplier_ID INT NOT NULL,
 Total_Amount INT NOT NULL,
 Buying_Date Date NOT NULL,
 PRIMARY KEY (Supplier_ID)
);

-- SITE ADMIN TABLE
CREATE TABLE Site_Admin
(
 Admin_Name VARCHAR(255) NOT NULL,
 Admin_ID INT NOT NULL,
 PRIMARY KEY (Admin_ID)
);

-- SERVICE PROVIDER TABLE
CREATE TABLE Service_Provider
(
 Provider_Name VARCHAR(255) NOT NULL,
 Service_Provided VARCHAR(255) NOT NULL,
 Provider_Id INT NOT NULL,
 PRIMARY KEY (Provider_Id)
);

-- MARKETING CAMPAIGN TABLE
CREATE TABLE Marketing_Campaign
(
 Start_Date Date NOT NULL,
 End_Date Date NOT NULL,
 Campaign_Name VARCHAR(255) NOT NULL,
```

```
  Campaign_Description VARCHAR(255) NOT NULL,
  Campaign_ID INT NOT NULL,
  Admin_ID INT NOT NULL,
  PRIMARY KEY (Campaign_Id)
);

-- MARKETING COSTS TABLE
CREATE TABLE Marketing_Costs
(
  Exp_amount INT NOT NULL,
  Cost_ID INT NOT NULL,
  Campaign_ID INT NOT NULL,
  PRIMARY KEY (Cost_ID)
);

-- SHOE TABLE
CREATE TABLE Shoe
(
  SKU INT NOT NULL,
  Shoe_Name VARCHAR(255) NOT NULL,
  Cost_Price INT NOT NULL,
  Shoe_Size INT NOT NULL,
  Selling_Price INT NOT NULL,
  Shoe_Brand VARCHAR(255) NOT NULL,
  is_sold NUMBER(1) NOT NULL,
  Post_Date Date NOT NULL,
  Quantity INT NOT NULL,
ADMIN_ID INT NOT NULL,
  PRIMARY KEY (SKU)
);

-- normalized SHOE ORDER table
CREATE TABLE Shoe_Order
(
  SKU INT NOT NULL,
  OrderID INT NOT NULL,
  PRIMARY KEY (SKU, OrderID)
```

```
);

-- normalized SHOE SUPPLIER table
CREATE TABLE Shoe_Supplier
(
 SKU INT NOT NULL,
 Supplier_ID INT NOT NULL,
 PRIMARY KEY (SKU, Supplier_ID)
);

-- CUSTOMER TABLE
CREATE TABLE Customer
(
 Customer_Name VARCHAR(255) NOT NULL,
 CustomerID INT NOT NULL,
 Customer_Number INT NOT NULL,
 Street_Address VARCHAR(255) NOT NULL,
 City VARCHAR(255) NOT NULL,
 Customer_Equity INT NOT NULL,
 Code_Name VARCHAR(255) NOT NULL,
 PRIMARY KEY (CustomerID)
);

-- SHOE ORDER TABLE
CREATE TABLE S_Order
(
 Order_Amount INT NOT NULL,
 OrderID INT NOT NULL,
 Quantity INT NOT NULL,
 Order_Date Date NOT NULL,
 CustomerID INT NOT NULL,
 Code_Name VARCHAR(255),
 Admin_ID INT NOT NULL,
 PRIMARY KEY (OrderID)
);

-- CODE TABLE
```

```
CREATE TABLE Code
(
  Code_Name VARCHAR(255) NOT NULL,
  Code_Uses INT NOT NULL,
  Total_Amount INT NOT NULL,
Admin_ID INT NOT NULL,
CustomerID INT NOT NULL,
  PRIMARY KEY (Code_Name)
);

--SERVICES TABLE
CREATE TABLE Shoe_Service
(
  Service_Cost INT NOT NULL,
  Service_ID INT NOT NULL,
  Service_Name VARCHAR(255) NOT NULL,
  Service_Date Date NOT NULL,
  SKU INT,
  PRIMARY KEY (Service_ID)
);

-- SHIPPING COST
CREATE TABLE Shipping_Cost
(
  ID INT NOT NULL,
  is_free NUMBER(1) NOT NULL,
  Shipping_Cost INT NOT NULL,
  OrderID INT NOT NULL,
  PRIMARY KEY (ID)
);

-- PROVIDES
CREATE TABLE Provides
(
  Provider_Id INT NOT NULL,
  Service_ID INT NOT NULL,
  PRIMARY KEY (Provider_Id, Service_ID)
```

```
);
-- QUERIES
-- Provides table foreign keys
ALTER TABLE Provides
  ADD FOREIGN KEY (Provider_Id) REFERENCES Service_Provider(Provider_Id);
ALTER TABLE Provides
  ADD FOREIGN KEY (Service_ID) REFERENCES Shoe_Service(Service_ID);

-- Shipping cost table foreign key
ALTER TABLE Shipping_Cost
  ADD FOREIGN KEY (OrderID) REFERENCES S_Order(OrderID);

-- Shoe_Service table foreign key
ALTER TABLE Shoe_Service
 ADD FOREIGN KEY (SKU) REFERENCES Shoe(SKU);

-- Marketing campaign foreign key
ALTER TABLE Marketing_Campaign
ADD FOREIGN KEY (Admin_ID) REFERENCES Site_Admin(Admin_ID);

-- Marketing Costs table foreign key
ALTER TABLE Marketing_CostS
ADD FOREIGN KEY (Campaign_ID) REFERENCES
Marketing_Campaign(Campaign_ID);

-- Shoe table foreign key
ALTER TABLE Shoe
ADD FOREIGN KEY (Admin_ID) REFERENCES Site_Admin(Admin_ID);

-- Code table foreign keys
ALTER TABLE Code
ADD FOREIGN KEY (Admin_ID) REFERENCES Site_Admin(Admin_ID);
ALTER TABLE Code
ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID);

-- Shoe order table foreign keys
ALTER TABLE S_Order
```

```
ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID);
ALTER TABLE S_Order
  ADD FOREIGN KEY (Code_Name) REFERENCES Code(Code_Name);
ALTER TABLE S_Order
  ADD FOREIGN KEY (Admin_ID) REFERENCES Site_Admin(Admin_ID);

-- Customer table foreign keys
ALTER TABLE Customer
  ADD FOREIGN KEY (Code_Name) REFERENCES Code(Code_Name);

-- Shoe Order table foreign keys
ALTER TABLE Shoe_Order
ADD FOREIGN KEY (SKU) REFERENCES Shoe(SKU);
ALTER TABLE Shoe_Order
ADD  FOREIGN KEY (OrderID) REFERENCES S_Order(OrderID);

-- Shoe Supplier table foreign keys
ALTER TABLE Shoe_Supplier
ADD  FOREIGN KEY (SKU) REFERENCES Shoe(SKU);
ALTER TABLE Shoe_Supplier
ADD  FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID);
```