Store Description

## Description:

The store design that we are basing our database on is our goto snack supplier “7-Eleven”. Most of our products revolve around food since that is the main purpose of our visits to the chain ourselves. Each product in our store has their own unique 7 digit UPC, “Universal Product Number”,  number, a Product Name, the Product Category they belong to, an amount of Stock left of the product, and finally an amount of the product that has been sold labeled “purchased”. Our store allows customers to view the products, create a cart, add items to their cart, remove items, and check out when ready. Our employees manage customers and products by adding, removing, and updating customer information, carts, and products, using multiple built in procedures that we have created in our database.

## Tables:

Products:

The store products are the most important part of our database and contain the most data values. We generated multiple product categories, brands, and amounts of stock, either current or purchased, that can be viewed at any time.

The different product categories that we offer are as follows:

- Pop
- Chips
- Candy
- Energy Drinks
- Hot Food
- Snacks
- Iced Coffee
- Gum
- Hot Beverages

With each of these categories we offer many different products across many brands. A few brands include:

- Pepsi
- Coke
- Starbucks
- Hershey’s
- Monster Energy
- Lays

Data for stock as well as purchased amount has been manufactured based on random generation of data for values 10 - 500. This allows us to query unique data amounts based on popularity of products and amount of stock remaining for our low stock trigger systems.

Customers:

Customers are the next table we created. Customers have their own Name fields as well as Customer Numbers, which are unique 5 digit numbers used to distinguish customer accounts.

Customers are able to make Carts for shopping, which are input as new tables that contain their customer ID for recognition purposes. These carts can then be “purchased”, which changes the

cart to a “completed” phase. Purchased carts are not removed as they can be referenced much like a receipt in order to see what has been purchased in the past.

Employees:
Employees have their own unique 5 digit employee numbers as well as their Names in the database. Employees are in charge of managing customers, carts, and products. They are able to add, remove, and update customers, products, and carts as needed in the database for accurate record keeping.

Alerts:
The alerts table is our response for interacting with Low Stock levels on products and allowing employees to add more product stock when needed. Every time an update is made to the “products” table, the Low Stock trigger reads the stock remaining and enters the product information into the Alerts table for the employees to see when stock is too low on an item.

## Database Access:

Employees:
Employees will be the only account that accesses the database in our design. Customers are only referenced as data in a table that has carts and purchases associated with them. Commands are sent to the database by Employees and used to work on all parts of the database.

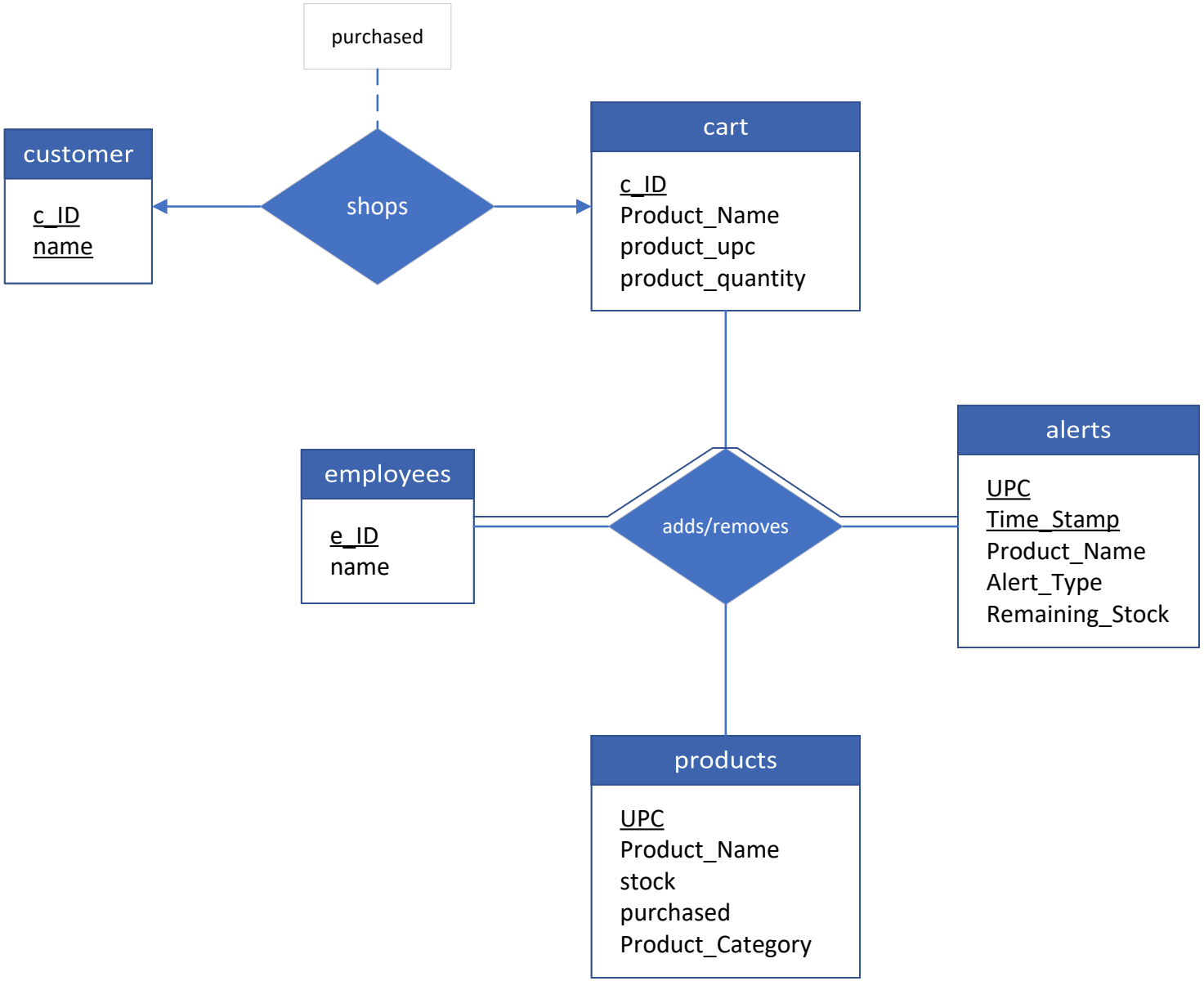
Guest:
A guest account has been created on the database with limited access. This is to simulate someone who would like to browse the products that our store offers but is unable to purchase anything. Guest accounts are only able to view the Products table in the database.

## Database Functions:

To complete everyday store functions in our database, we have created functions in MySQL called “Stored Procedures”. These procedures work much like preconfigured functions or scripts inside our database. Below is a list of each procedure’s name and a short description of how they operate.

- AddCustomer
    - Adds customers to the database with a preset name and unique customer ID
- CreateCart
    - Creates a new table, ‘cart’, for a customer to use based on their customer ID
- AddToCart
    - Starts a transaction that adds products to the cart table while removing the same amount of stock of that product from the store.
- RemoveFromCart
    - Starts a transaction that removes a product from the customer cart while updating the stock of the product back into the products table
- PurchaseCart

    - Updates the cart name to indicate that it has been purchased and no longer useable
- DeleteCart
    - Removes a cart from the database
- DeleteCustomer
    - Removes a customer from customer table
- AddStock
    - Adds a new product to the database, updating the current stock
- AddProduct
    - Adds a new product to the database to be searchable by customers
- DeleteProduct
    - Removes products from the database
- Top20Products
    - Lists the top products that have been purchased in the database
- Top3Categories
    - Lists the top products that have been purchased in the database by product category


purchased
customer
c_ID
name
shops
cart
c_ID
Product_Name
product_upc
product_quantity
employees
e_ID
name
adds/removes
alerts
UPC
Time_Stamp
Product_Name
Alert_Type
Remaining_Stock
products
UPC
Product_Name
stock
purchased
Product_Category

Team Contributions

**Team Members:**
Cody Holoday
Darren Hutchinson
Chris Liberman
Ian Boulis

**Database Design:**
Darren Hutchinson
Cody Holoday
Chris Liberman
Ian Boulis

**Database Hosting:**
Cody Holoday

**Database Setup:**
Cody Holoday
Darren Hutchinson

**DataPopulation:**
Cody Holoday
Darren Hutchinson

**File Gathering and Completion:**
Chris Liberman
Ian Boulis
Cody Holoday
Darren Hutchinson

**Function Creation:**
Darren Hutchinson
Ian Boulis

**Function / Query Testing:**
Chris Liberman
Ian Boulis

**Trigger Creation and Testing:**
Chris Liberman

**ER Diagram Creation:**
Chris Liberman

**Video Creation:**
Cody Holoday
Ian Boulis

**Database Description:**
Darren Hutchinson

Sample Queries

```
2 •   CALL Top20Products();
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conten

| UPC | Product_Name | stock | purchased | Product_Category |
|---|---|---|---|---|
| 1255307 | Reese's pieces | 306 | 244 | Candy |
| 1962907 | Cherry Coke | 60 | 240 | Pop |
| 1256161 | Doritos Nacho | 456 | 230 | Chips |
| 1221306 | Diet Cherry Coke | 495 | 226 | Pop |
| 1234630 | Doritos Cool Ranch | 181 | 226 | Chips |
| 1400828 | Lays Cheddar | 396 | 216 | Chips |
| 1041628 | Monster Ultra | 77 | 214 | Energy Drinks |
| 1104280 | Cheeseitz | 349 | 201 | Snacks |
| 1990653 | Mtn Dew Voltage | 35 | 194 | Pop |
| 1244478 | 5_Gum Spearmint | 84 | 194 | Gum |
| 1468478 | Starbucks Double ... | 202 | 193 | Iced Coffee |
| 1176374 | Starbucks Triple V... | 19 | 192 | Iced Coffee |
| 1186015 | Monster | 65 | 188 | Energy Drinks |
| 1677641 | Lays Ripples | 194 | 187 | Chips |
| 1552156 | Starbucks Double ... | 361 | 186 | Iced Coffee |
| 1805969 | Lays original | 412 | 179 | Chips |
| 1268761 | Kitkat | 491 | 171 | Candy |
| 1003102 | Cherry Pepsi | 268 | 167 | Pop |
| 1011038 | Hersheys | 334 | 150 | Candy |
| 1915675 | Nilla Wafers | 452 | 148 | Snacks |


```
2 •   CALL CreateCart('Todd Arney');
```

Sample Queries

```
CALL Top3Categories();
```

Result Grid | Filter Rows:

| Product_Category |
| --- |
| Pop |
| Candy |
| Energy Drinks |

# Challenges

One of the main challenges that we encountered was the generation of random data for employees, customers, and products. We used a random number generator to create Employee ID numbers, Customer ID numbers, and Product UPCs.

Another challenge was figuring out how triggers are created and function inside a MySQL database. Using the MySQL reference manual to teach ourselves about them we were able to figure out how to get an alert when products are below 50 stock. This alert is triggered every time there is an update to the products table, this is done to reduce the size of the alerts table that is created, while still giving accurate information.

Another challenge was figuring out how procedures are created and are called. Using the MySQL reference manual to teach ourselves about them we were able to figure out how to create procedures for the required database functions. I.E., CreateCart, PurchaseCart, CreateCustomer, AddProduct, etc.

For our project, we didn't end up using programming languages like python or java. We accomplished it exclusively in MySQL. Doing it this way allowed us to create the procedures right on the database without affecting each other with testing or needing to share source code files to help each other. We also hosted a MySQL server that we could all individually access on separate computers so that everything didn’t have to be done on a single computer.