Joshua Kim | Thong Thomas Nguyen

Team BreadHeads Group 48

Title: Global Bread Co.

CS 340

Project Step 6 Final

December 10, 2023

**URL:** [**http://flip4.engr.oregonstate.edu:54920/**](http://flip4.engr.oregonstate.edu:54920/)

#### **Executive Summary**

From planning to deployment, there have been numerous changes to bring the project up to standard. Firstly, the overview was upgraded with a more specified and numerical mission statement. Entity attributes were swapped to camelCase. To advocate the significance of allergens in the bread products as part of the marketing for Global Bread Co., the intersection table, “allergensProduct”, was kept as a feature on the “allergens” page. The same decision was applied to the “soldProducts” intersection table, as it was placed in the “sales” page. With the decision of having the “customers” address being expanded—street, city, zip code, normalization was ultimately configured to be 2NF.

Initially, it was a struggle to implement same page entity updates for the UI, but edit pages were eventually used for efficiency and ease of use. The user interface in the update portions were also updated with dynamic select dropdowns for foreign keys. To streamline usability, the line totals for “soldProducts” were automatically calculated by accessing the product’s unit price. Stock, an attribute from “breadProducts”, was removed due to its changing nature and complexity when adding, updating, or removing a sale. The DML.sql and app.py functions were annotated with the equivalent queries, and decimal-type attributes were swapped to decimal(10, 2). The default decimal did not show floating point. ProductID in “soldProducts” was shifted to NULL to ensure that a sale remains after deleting the product. The same foreign key, ProductID, was also set to the ON DELETE SET NULL cascade. A large portion of the project was dedicated to troubleshooting the database connection to the server. Fortunately, this was fixed by creating a helper function for the database connection right before any query was executed. After the application was functional, Bootstrap and CSS were applied to keep the website cohesive and aesthetic with warm and inviting colors.

#### 

#### 

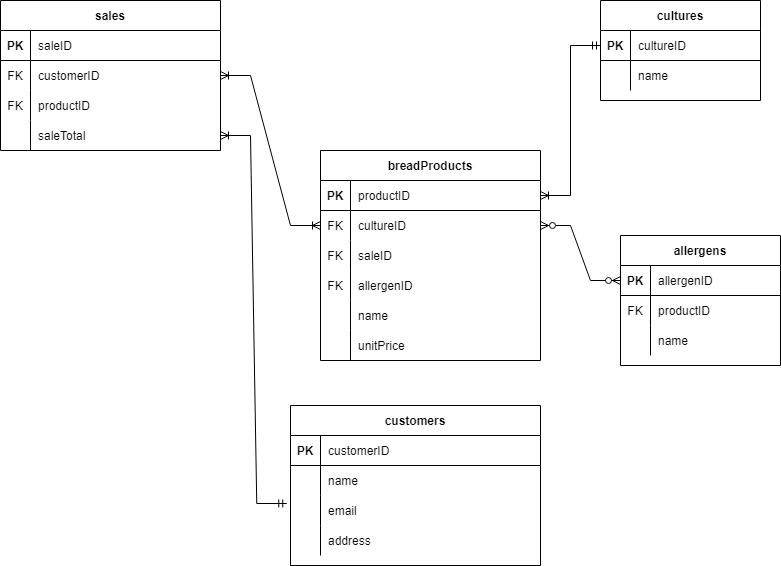
#### **Overview**

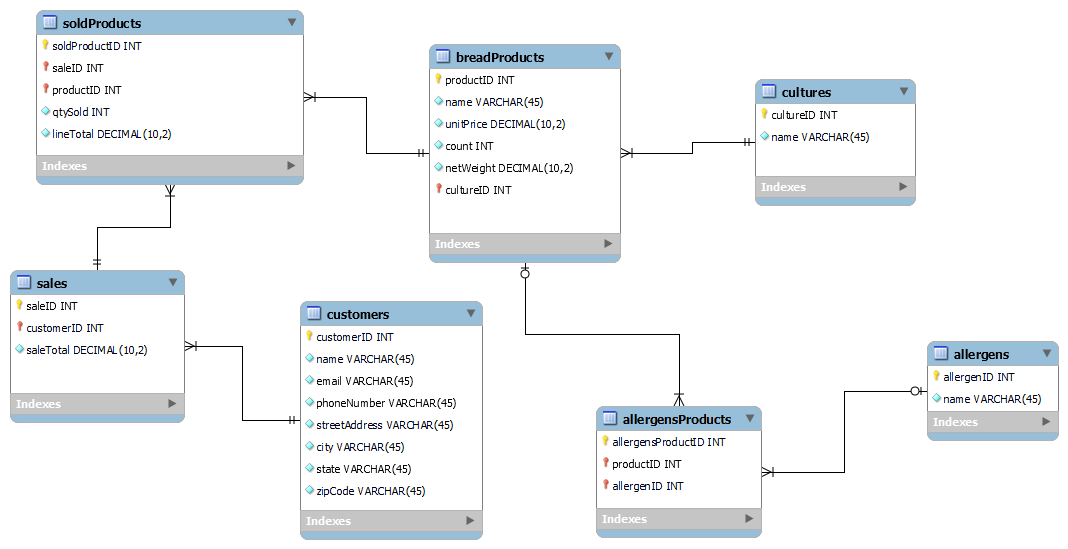
Global Bread Co. is predicted to make $500,000 this year and is hoping to have at least 1,000 sales this year by selling at least 60 different types of bread. Whether it be from wholesale buyers or curious, bread enthusiasts, Global Bread Co. seeks to solve their customers’ need for sourcing different breads from different cultures. With such an ambitious goal in mind, the company will need to keep track of their both their customer’s information and the sales that they will make. As the company is trying to sell virtually every type of bread in the world, Global Bread Co. wishes to keep track of the cultural origins and allergens of their products to incentivize customers to try something new. The database will record *Sales* of *BreadProducts (Cultures*) to *Customers*, and will keep track of *Allergens* in their *BreadProducts*.

#### **b) Database Outline, in Words**

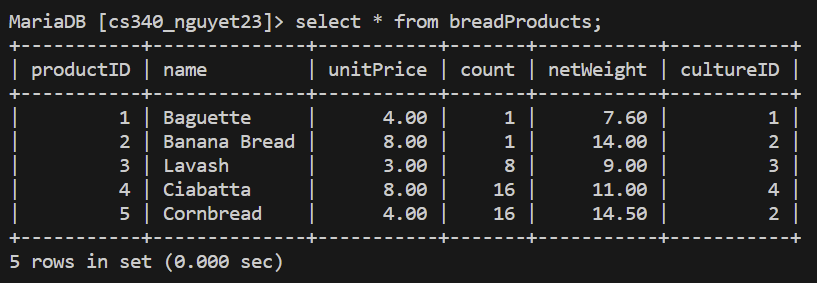
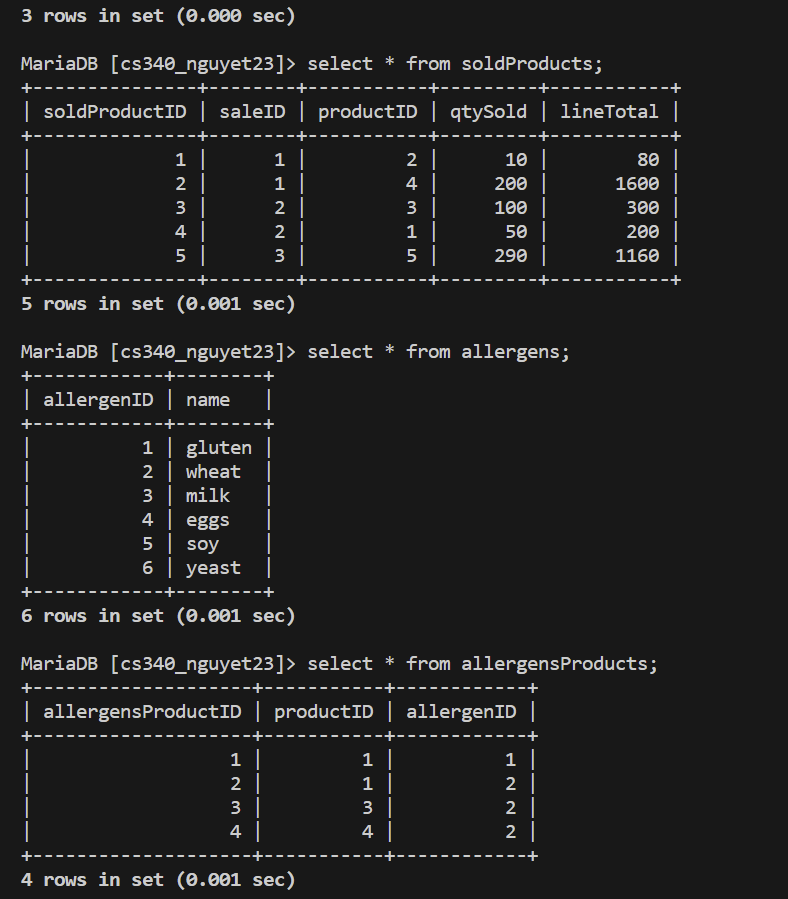
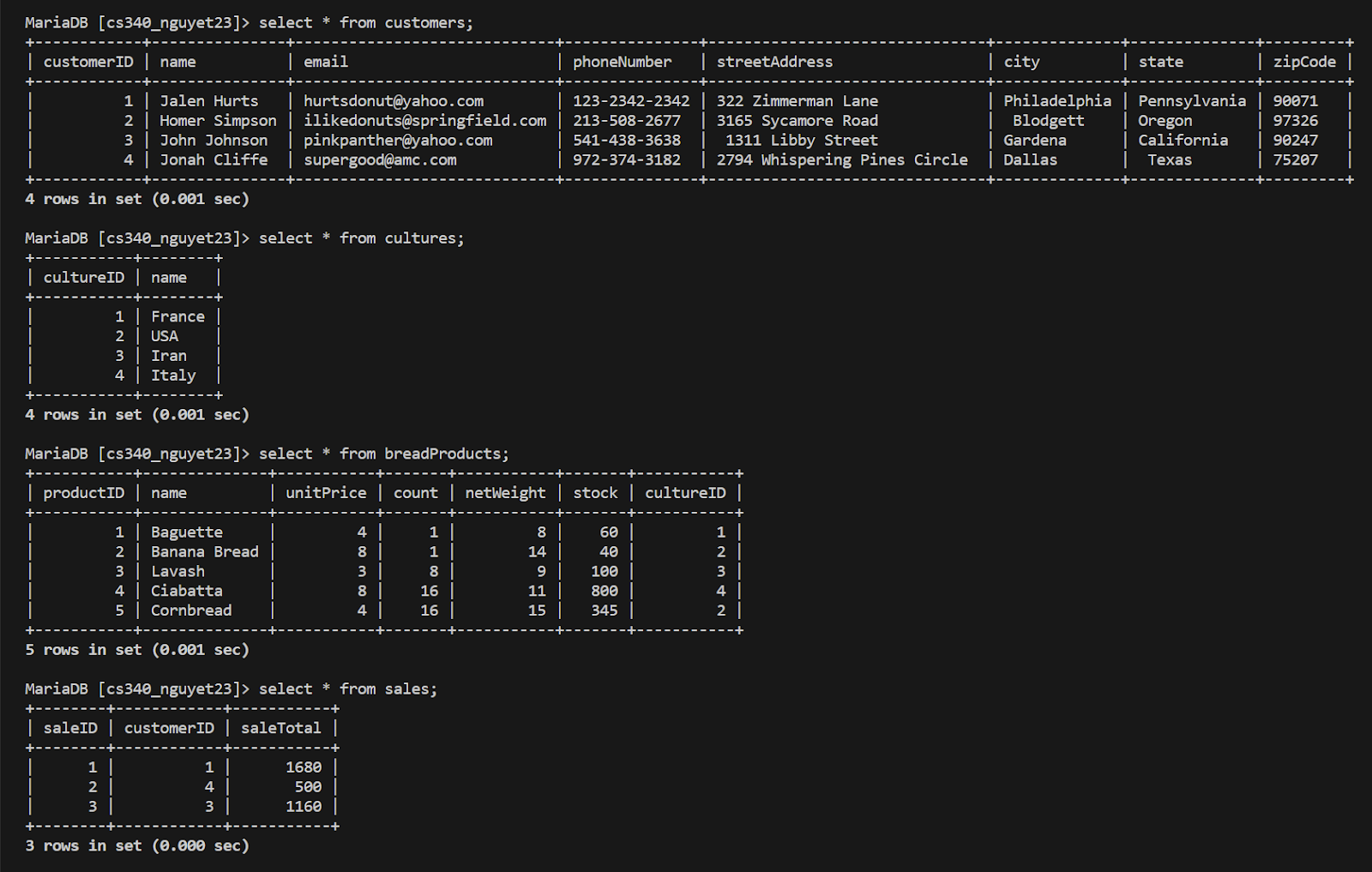
* **customers**: records the details of Customers we do business with
  + customerID **pk**: int, auto\_increment, unique, not NULL, PK
  + name: varchar(45), not NULL
  + email: varchar(45), not NULL
  + phoneNumber: varchar(45), not NULL
  + streetAddress: varchar(45), not NULL
  + city: varchar(45), not NULL
  + state: varchar(45), not NULL
  + zipCode: varchar(45), not NULL
  + Relationship: a 1:M relationship between customers and sales is implemented with customerID as a FK inside of sales
* **breadProducts**: records the details of all the breads we sell
  + productID **pk**: int, auto\_increment, unique, not NULL, PK
  + cultureID **fk**: int, not NULL, FK
  + name: varchar(45), not NULL
  + unitPrice: decimal(10,2), not NULL
  + count: int, not NULL
  + netWeight: decimal(10,2), not Null
  + Relationship: a M:M relationship between breadProducts and sales as soldProducts (Joint Table)
  + Relationship: a M:M relationship between breadProducts and allergens as allergensProducts (Intersection Table)
  + Relationship: a M:1 relationship between breadProducts and cultures is implemented with a cultureID as a FK inside of breadProducts
* **sales**: records the details of sales made from customers (ex. Cart orders)
  + saleID **pk**: int, auto\_increment, unique, not NULL, PK
  + customerID **fk**: int, not NULL, FK
  + saleTotal: decimal(10,2), not NULL
  + Relationship: a M:M relationship between breadProducts and sales is implemented as soldProducts (Intersection Table)
  + Relationship: a M:1 relationship between sales and customers is implemented with customerID as a FK inside of sales
* **soldProducts (Intersection Table):** records the details of “individual product-sales” from customers
  + soldProductID **pk**: int, auto\_increment, unique, not NULL, PK
  + saleID **fk**: int, not NULL, FK
  + productID **fk**: int, NULL, FK
  + qtySold: int, not NULL
  + lineTotal: decimal(10,2), not NULL
* **cultures**: categorizes all the breads originating from a certain region
  + cultureID **pk**: int, auto\_increment, unique, not NULL, PK
  + name: varchar(45), not NULL
  + Relationship: a 1:M relationship between cultures and breadProducts is implemented with cultureID as a FK inside of breadProducts
* **allergens**: categorizes major food allergens
  + allergenID **pk**: int, auto\_increment, unique, not NULL, PK
  + name: varchar(45), not NULL
  + Relationship: a M:M relationship between allergens and breadProducts is implemented as allergensProducts (Intersection Table)
* **allergensProducts (Intersection Table)**: records the allergens present in bread products
  + allergensProductID **pk**: int, auto\_increment, unique, not NULL, PK
  + allergenID **fk**: int, not NULL, FK
  + productID **fk**: int, not NULL, FK

#### **Entity-Relationship Diagram**



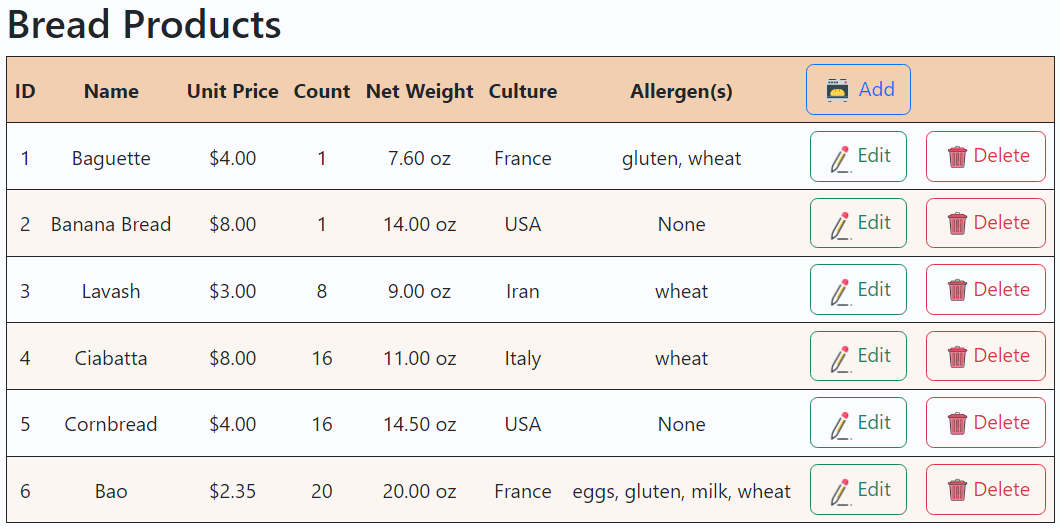
**Schema**

#### **Example Data**

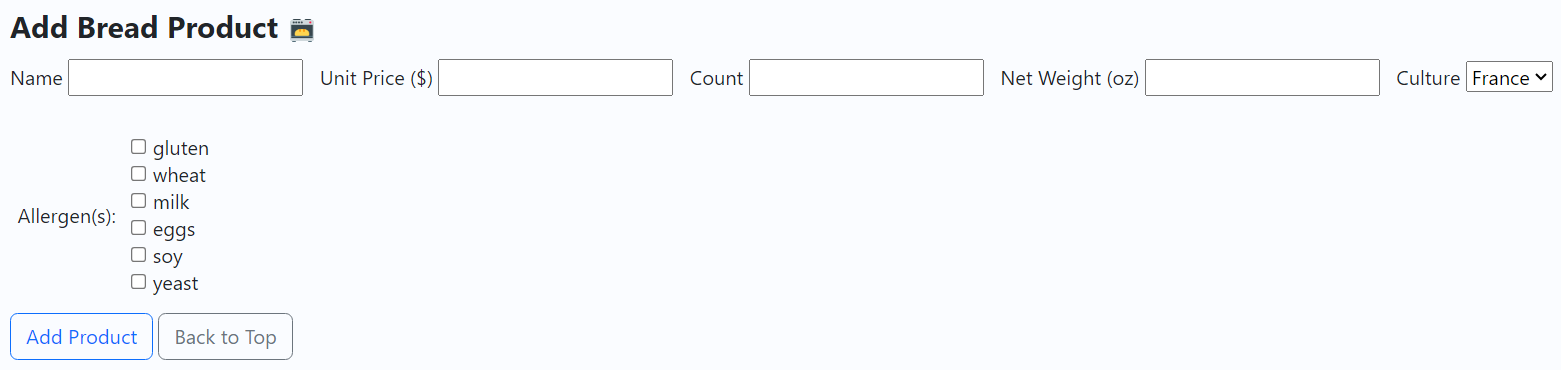


**User Interface Screenshots**

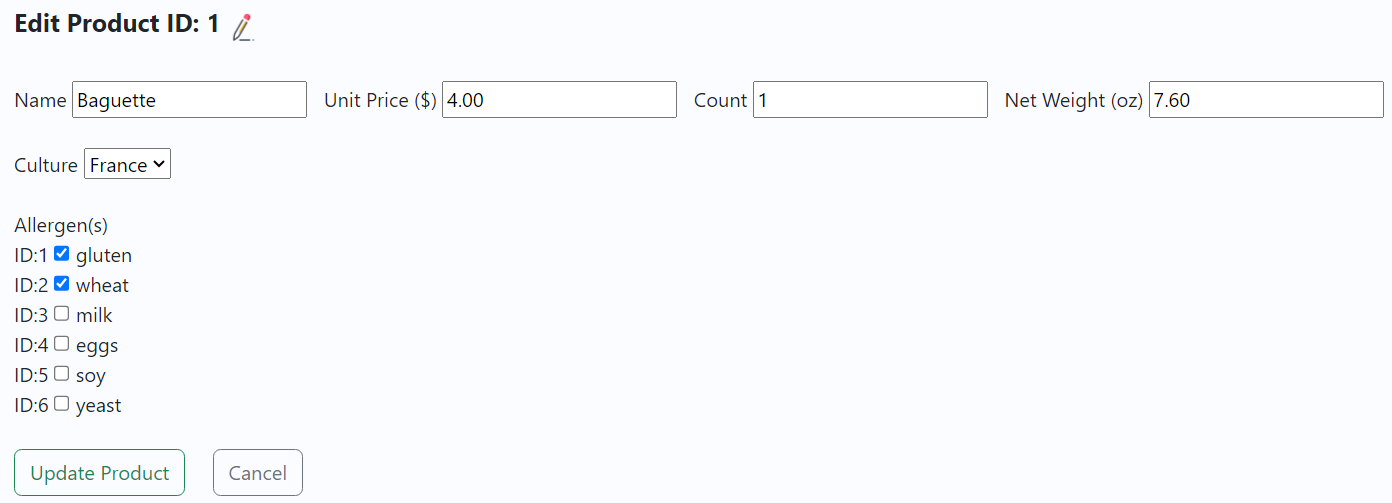
**[Bread Products]:** READ/DISPLAY and DELETE



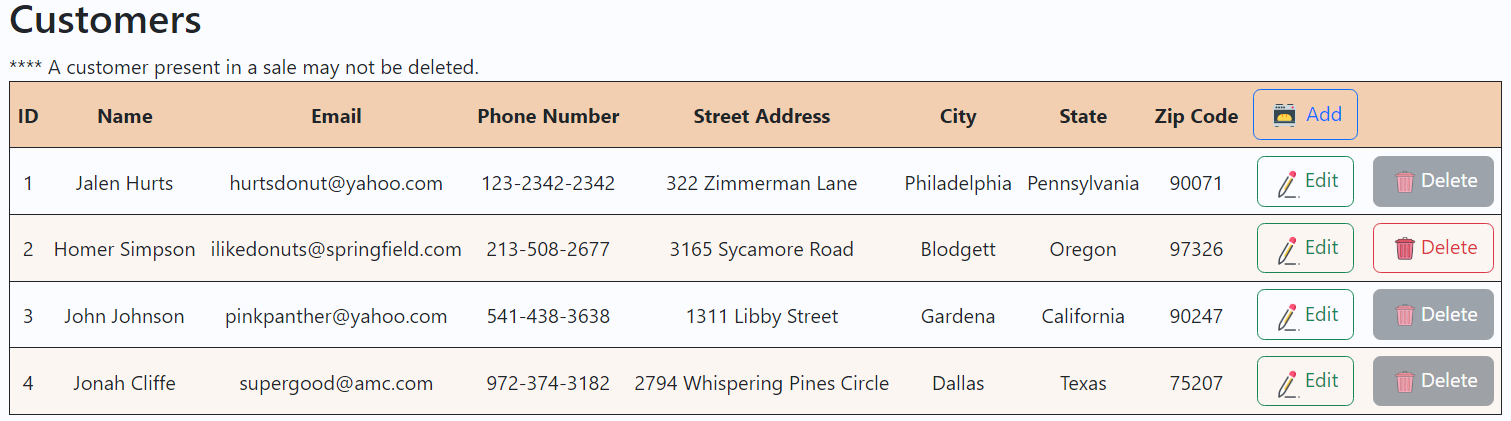
**[Bread Products]:** CREATE/ADD



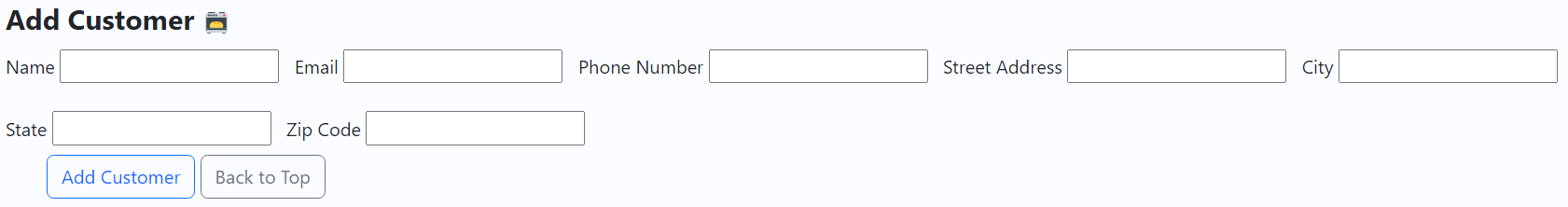
**[Bread Products]:** UPDATE/EDIT  
\*\* After pressing EDIT on the Table



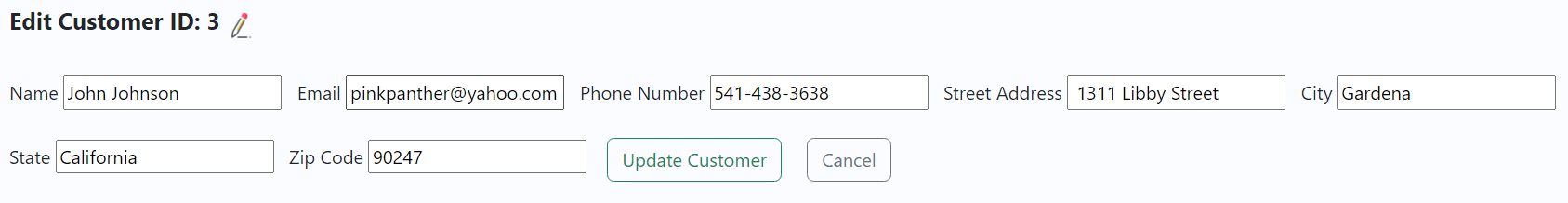
**[Customers]:** READ/DISPLAY and DELETE



**[Customers]:** CREATE/ADD

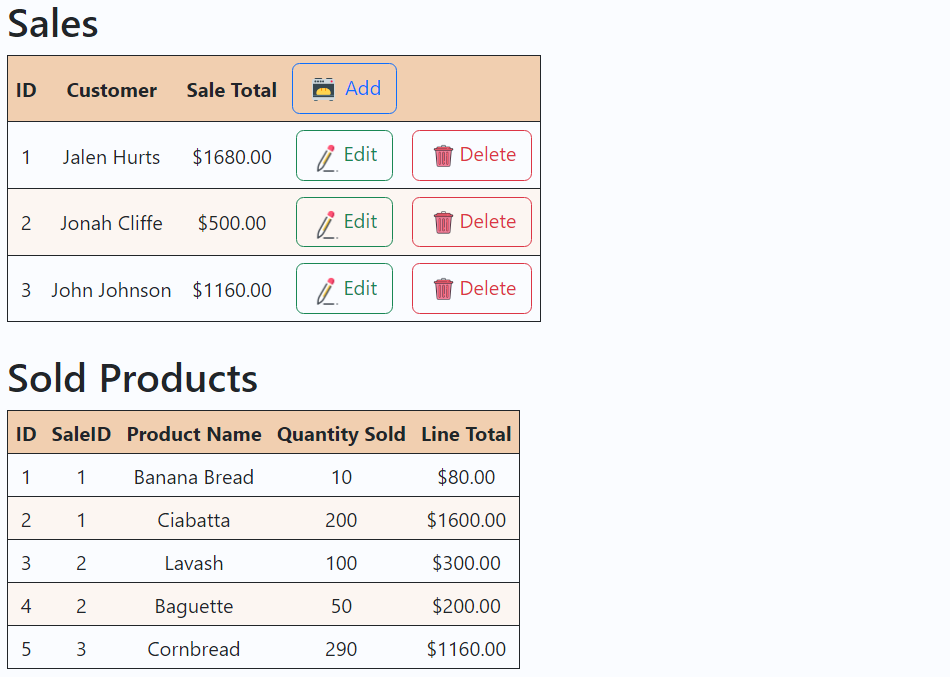


**[Customers]:** UPDATE/EDIT



**[Sales(1:M) & SoldProducts(M:N)]:** READ/DISPLAY and DELETE

\*\* When a sale is deleted, the soldProducts related to that sale are also deleted.

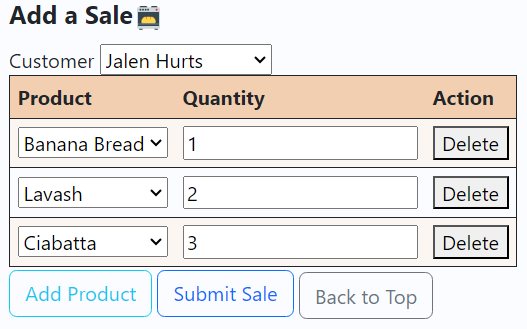


**[Sales(1:M) & SoldProducts(M:N)]:** CREATE/ADD

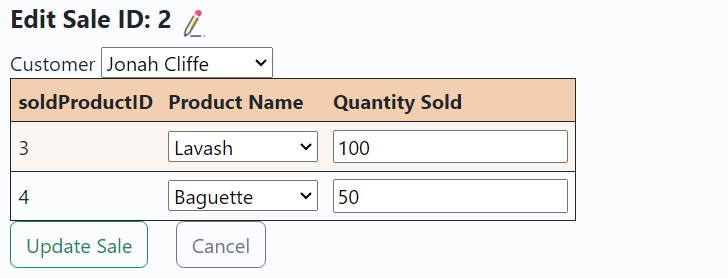
\*\* Every added product will be added to soldProducts

\*\* Submit Sale will add the Sale.

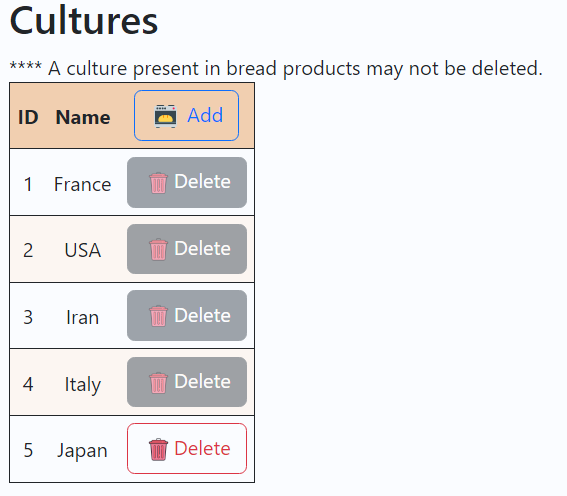
\*\* lineTotal from soldProducts is auto-calculated.



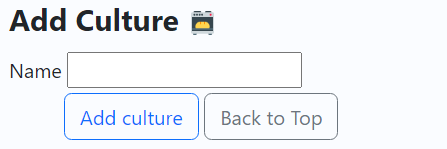
**[Sales(1:M) & SoldProducts(M:N)]:** UPDATE/EDIT



**[Cultures]:** READ/DISPLAY and DELETE

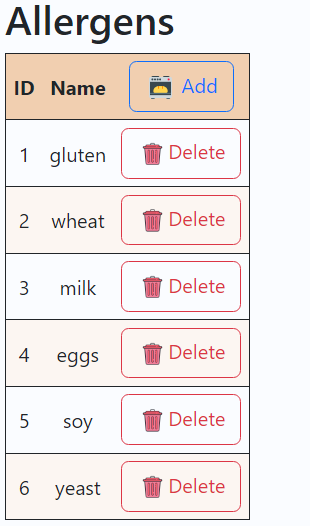


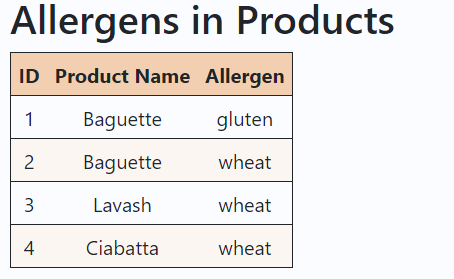
**[Cultures]:** CREATE/ADD



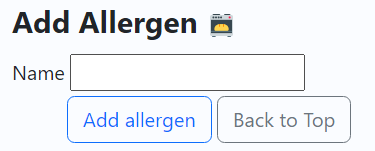
**[Allergens (1:M) and allergensProducts(M:N)]:** READ/DISPLAY and DELETE

\*\* When an allergen is deleted, the allergensProducts related to that allergen are also deleted.



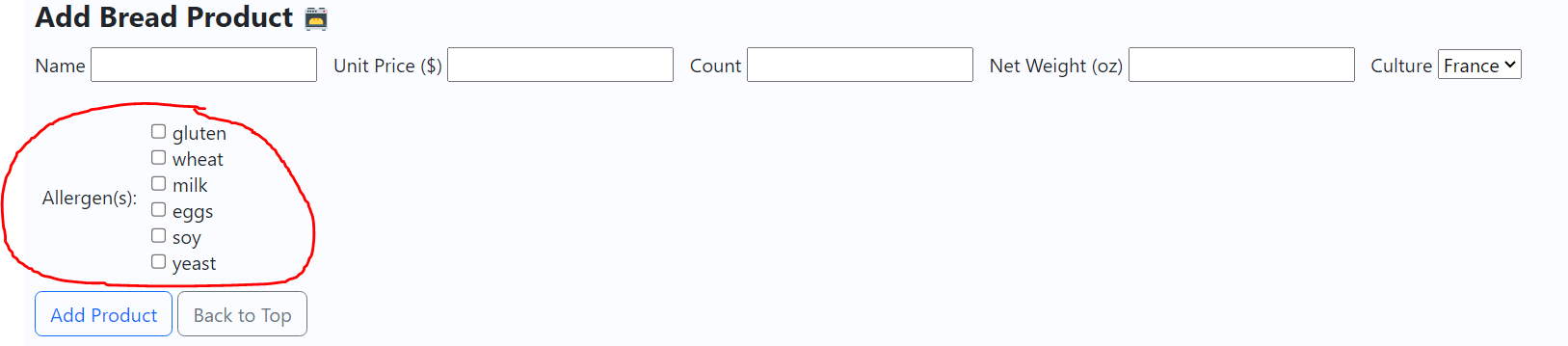
****

**[Allergens]:** CREATE/ADD



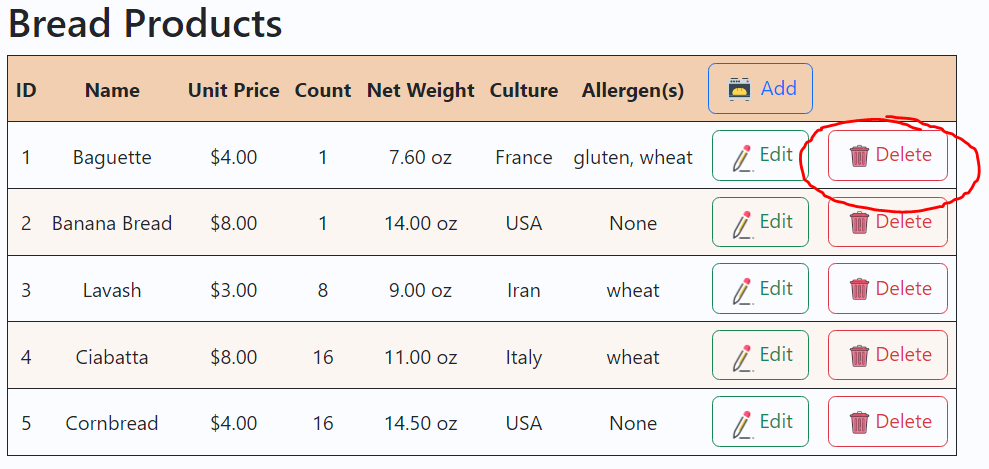
**[AllergensProducts (M:N)]:** CREATE/ADD

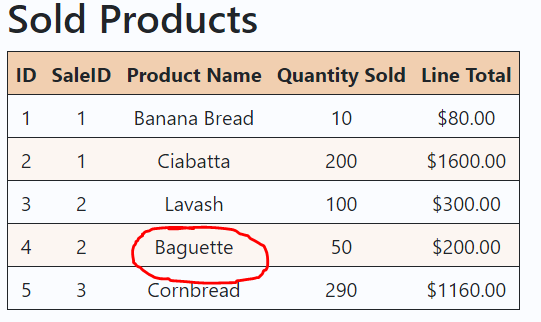
\*\* Back in adding breadProducts, when an allergen is checked, the allergens in this product will be added to allergensProducts.

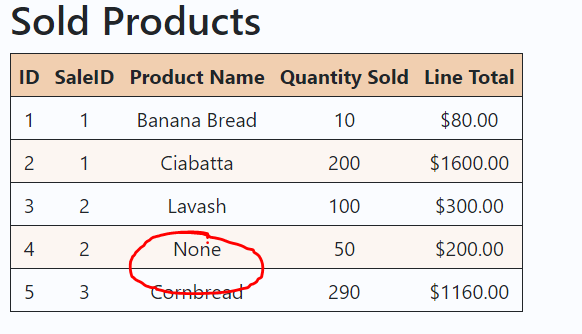


[soldProducts(M:N)]: DELETE TO NULL

\*\* Delete breadProduct to set FK in soldProducts to NULL







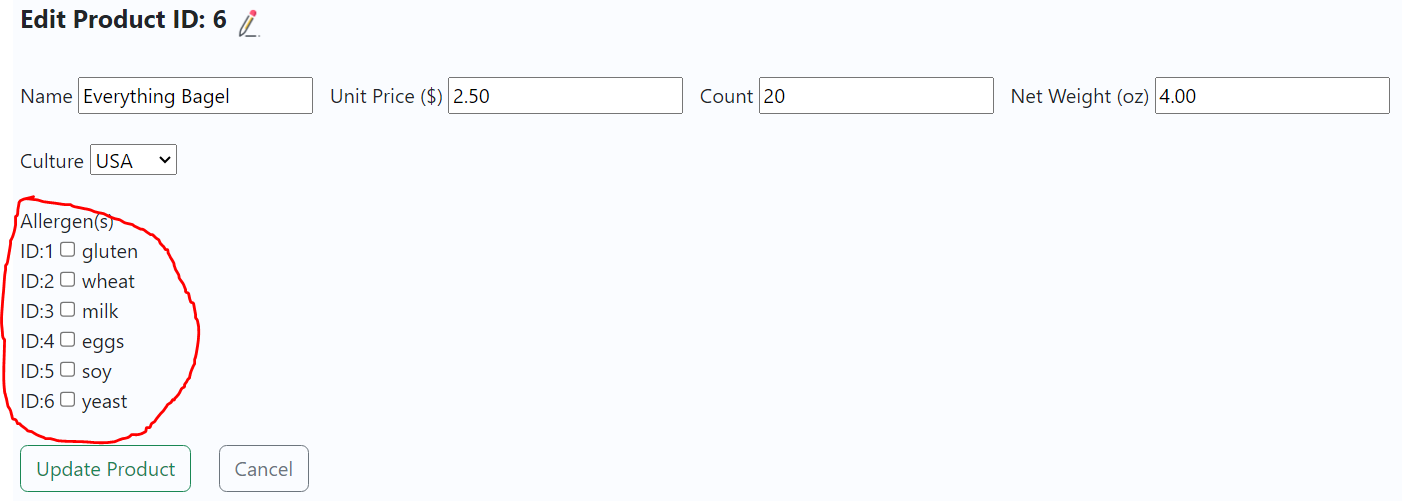
[allergens to breadProducts (connected by allergensProducts intersection)]:

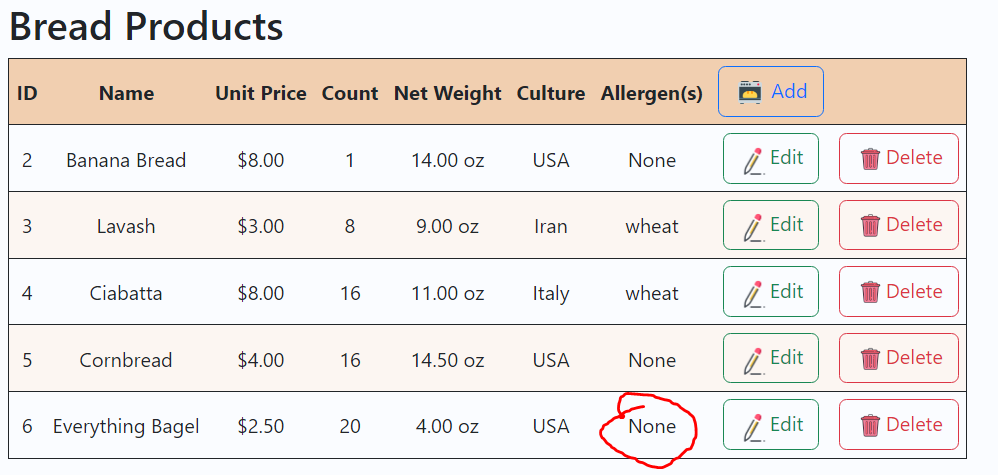
UPDATE TO NULL

\*\* Can update breadProducts allergens to None (checkbox)

\*\* Can delete allergen in allergens table to set breadProducts allergens to None







[allergens to breadProducts (connected by allergensProducts intersection)]:

DELETE TO NULL

\*\* Can delete allergen in allergens table to set breadProducts allergens to None



