Team Members: Sean Bleyl, Joshua Griep

URL: <http://classwork.engr.oregonstate.edu:5420/>

*Executive Summary*

FurniTech is a growing furniture company with four retail locations located in Los Angeles, New York, Chicago, and San Francisco. The company sells a wide variety of furniture products, including both stock and custom-made items. Each year, FurniTech sells approximately 2,000 items and generates around $1.2 million in revenue across all locations.

Each location has approximately 3-5 employees in various roles running each store, such as sales representatives, stockers, furniture designers, and managers. With plans to expand in the next 3 years, the company intends to implement a database-driven website to better track their assets and make informed business decisions. The database will also have a record of what orders were placed by which customers, what furniture type has been purchased, as well as the sales rep that helped close the transaction, and the location the orders were placed at.

From the initial database design in Step 1 to the final implementation of DDL.sql, PL.sql, and the UI, our tables and ERD stayed aligned with the original outline. The only change was adding seed data to PL.sql and DDL.sql so each location shows at least one product based on Step 4 feedback.

In Step 5 I planned to use stored procedures such as our ‘Delete Leroy Jenkins’ and ‘Reset’ implemented in Step 4 for the CRUD. After many attempts and continuously receiving ‘404 cannot POST’ errors, and not being able to get the Create, Update, and Delete to work, I switched to implement the CRUD routes directly in my Node/Express code using inline SQL queries rather than calling stored procedures. When this also failed, I then decided to try changing the PORT to ‘5420’, and the forms worked immediately.

As far as AI tools, I used AI throughout the project to ask general questions regarding implementation, especially in Part 5 as I implemented a method that was outside of the scope I originally anticipated. I used only ChatGPT. As far as how I used this information, I find that ChatGPT is typically better used to debug my code or give me guidance on how to implement certain code to make it work for my project. I find that whenever it attempts to generate code, it usually has several issues or decides to implement something entirely different.

*Project and Database Outlines*

**Customers:** records the details of Customers we do business with. Information on their basic information and contact information.

* customerID: int, auto\_increment, unique, not NULL, PK
* firstName: varchar(25), not NULL
* lastName: varchar(25), not NULL
* phone: varchar(10), not NULL
* email: varchar(30), not NULL
* Relationship:
  + 1:M between Customers and Orders. A customer can place one or more orders.

**Products:** records all product types sold at FurniTech, including whether they are stock or custom items.

* productID: int, auto\_increment, unique, not NULL, PK
* productName: varchar(50), not NULL
* productType: varchar(15), not NULL (Custom, Stock)
* unitPrice: float(19,2), not Null
* Relationship:
  + 1:M with ProductLocation. Product can be available at multiple locations
  + 1:M with ProductsOrdered. Product can appear in many orders

**Employees:** records employees that work at FurniTech

* employeeID: int, auto\_increment, unique, not NULL, PK
* firstName: varchar(25), not NULL
* lastName: varchar(25), not NULL
* phone: varchar(10), not NULL
* email: varchar(30), not NULL
* employeeAddress: varchar(50), not NULL
* employeeCity: varchar(25), not NULL
* employeeStateAbbr: varchar(2), not NULL
* locationID: int, not NULL, FK
* ordersFulfilledCount: int, NULL
* ordersActiveCount: int, NULL (Assigned active orders)
* employeeRole: varchar(10), not NULL (Sales, Stocking, Designer, Manager)
* Relationship:
  + M:1 between Employees and Locations. Many employees can belong to a single location
  + 1:M between Employees and Orders. An employee can fulfill multiple orders

**Locations:** records details of each FurniTech location.

* locationID: int, auto\_increment, unique, not NULL, PK
* locationName: varchar(25), not NULL
* locationAddress: varchar(50), not NULL
* locationCity: varchar(25), not NULL
* locationStateAbbr varchar(2), not NULL
* phone: varchar(10), not NULL
* Relationship:
  + 1:M with Employees. Location can employ multiple staff members.
  + 1:M with Orders. Can fulfill multiple orders.
  + 1:M with ProductLocation. Locations can carry multiple products

**Orders:** records of orders placed by the Customers. Invoices.

* orderID: int, auto\_increment, unique, not NULL, PK
* customerID: int, not NULL, FK
* employeeID: int, NULL, FK (Orders entered but not yet assigned to a sales rep)
* locationID: int, not NULL, FK
* address: varchar(100), not NULL
* dateOrdered: date, not NULL
* dateEstimateDelivery, date, not NULL
* dateDelivered, date, NULL
* orderStatus: varchar(10), not NULL (Pending, In-Transit, Delivered, Cancelled)
* subtotal: decimal(8, 2), not NULL
* tax: decimal(5, 2), not NULL
* orderTotal: decimal(8, 2), not NULL
* Relationship:
  + M:1 with Customers, Employees, and Locations. Each order linked to one customer, employee, and location.
  + 1:M with ProductsOrdered. Each order can contain multiple products.

**ProductsOrdered**: records what products the customer would like to order. Each customer can have an order more than one product, each product will have it’s own ProductsOrderedID and will be totaled for orders.

* orderItemID: int, auto\_increment, unique, not NULL, PK
* orderID: int, not NULL, FK
* productID: int, not NULL, FK
* quantity: int, not NULL
* productPrice: decimal(8,2), not NULL (Price of individual product)
* totalProductPrice: decimal(8,2), not NULL (Takes account price of product including quantity)
* Relationship:
  + M:1 with Orders. Each ProductOrdered tied to specific order.
  + M:1 with Products. Each ProductOrdered tied to one product in an order.
  + This forms a bridge table for the M:M relationship between Orders and Products. One order can include many products, and one product can appear in many orders.

**ProductLocation:** record of what products are available at each location. This allows us to track per-location inventory and ensure accurate availability by location. Intersection between Products and Locations.

* productLocationID: int, auto\_increment, unique, not NULL, PK
* locationID: int, not NULL, FK
* productID: int, not NULL, FK
* Relationship:
  + M:M between Locations and Products, resolved by this table. A product can be stocked at many locations, and each location can carry many products.

*Entity-Relationship Diagram*

*A diagram of a server

AI-generated content may be incorrect.*

*Schema*

A screenshot of a computer

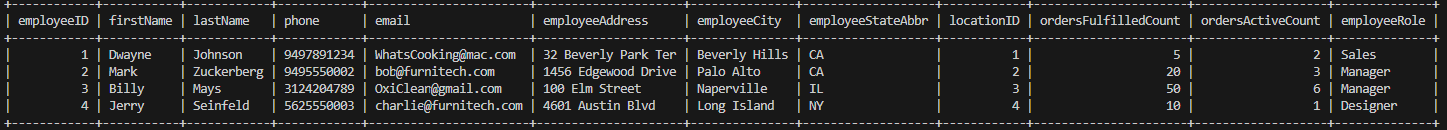
AI-generated content may be incorrect.

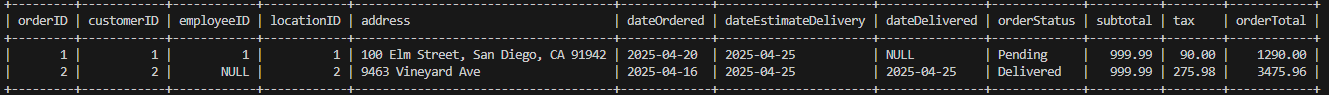
*Sample Data*

Customers:

A screenshot of a computer screen

AI-generated content may be incorrect.

Employees:

Orders:

Products:  
A black screen with white text

AI-generated content may be incorrect.

ProductsOrdered:

A black background with white text

AI-generated content may be incorrect.

Locations:  
A black and white screen with white text

AI-generated content may be incorrect.

ProductLocation:

A screen shot of a computer

AI-generated content may be incorrect.

*UI Screen Shots with Informative Titles*

Home:

Customers:

Products:

Locations:

Employees:

Orders:

Order Details:

Inventory:

*Citations*

All work is original and our own.