

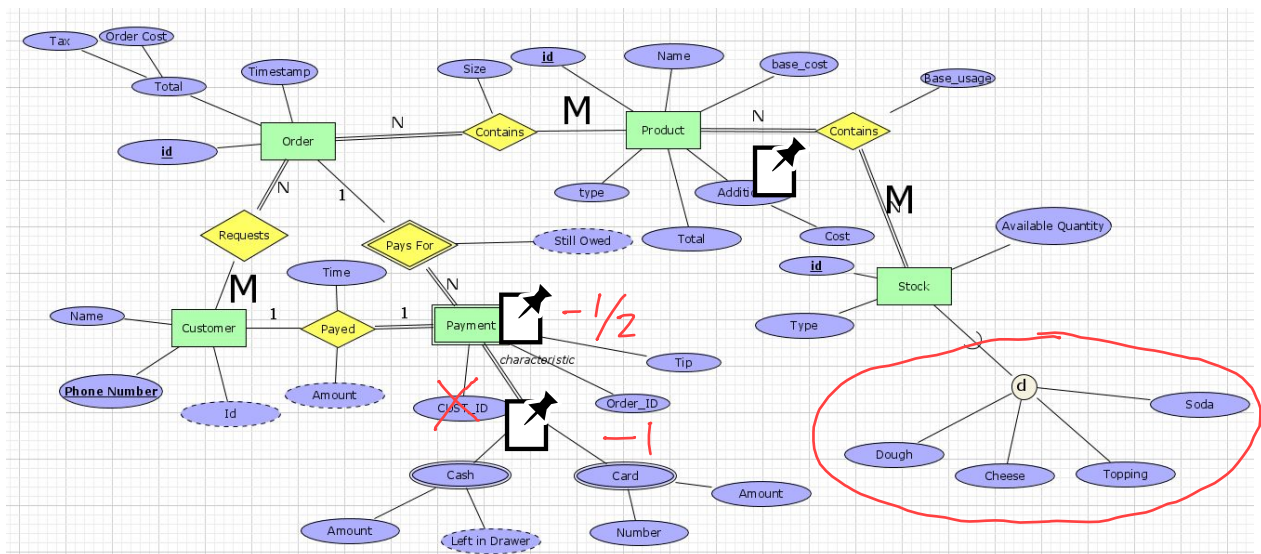
91.5 / 100

Phase 1: Conceptual Database Design

Problem Statement

The application we're proposing to build is a Point of Sale System for a pizzeria that will be used by the cashier. A database is required for this system since it will need to store and access information about the pizzas and other food available, their costs, the inventory & customer information for a faster and personal experience.

Conceptual Database Design



Entity and Relationship Explanation

The order entity represents the table of all orders. It will contain a unique id derived from the ordering customer's phone number and the order number for that customer, incremented each time a customer with that phone number orders. It will also have a timestamp attribute and hold the total cost of the order, the cost for the food itself, and the tax.

It is related to the customer who requests the order. The customer will have a name and a unique id from their phone number.

-1/2

The customer is related to a payment through the amount they pay for the order. The payment entity holds the tip paid. It has a the option for the payment to be paid with cash, card or a combination. The cash entity holds the amount of cash and a cash left in drawer attribute. The card option has a unique id which is the card number and holds the amount paid on the card. This works for gift or credit cards. Payment is then related back to the order that entity by the amount still owed on the order, derived from payments paid on one order. We assume that the tax rate is 2.6% (Tax rate in Missouri); this can be updated in the configuration page. - 1/2

The order entity contains products held in the product entity, specified by a name and unique id for the product. The product in the order also holds the type of product and the base cost of each product, say, a small, plain pizza. All products come in 3 size, small, medium and large. The base cost is multiplied by '1.5' to find the cost of the item if it's size was medium or by '2' if it's size was large. The product can then have additions, such as a extra toppings

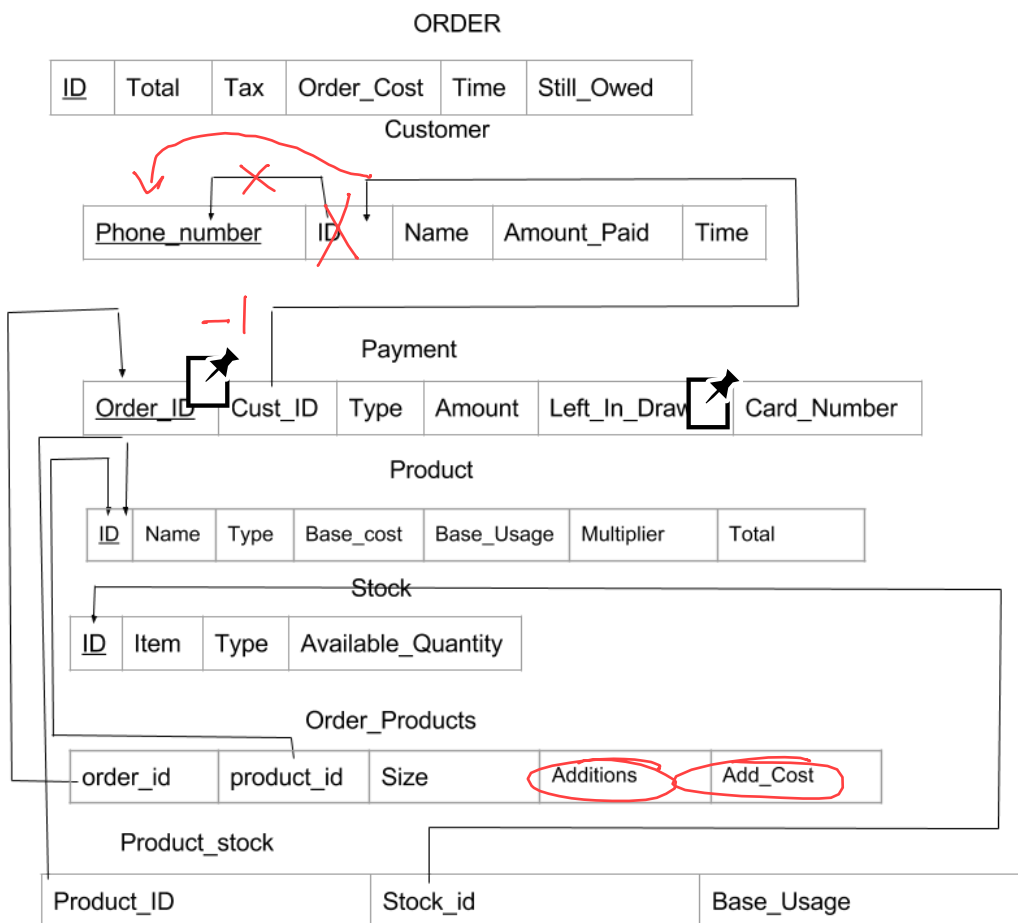
Products are related to the stock the product uses by a base usage, used to denote how much is used in a base product. For example, a small pepperoni pizza would use '1' dough, '1' cheese, and '1' pepperoni, while a medium would have a multiplier of 1.5, '1.5' dough, '1.5' cheese, and '1.5' pepperoni. This is used to update stock, which has a 4 disjointed options with all with types, a unique id used for ordering, and the available quantity of the stock item measured in the base usage of each stock item.

Functional requirements

- Identify a customer from their phone number with information such as their name and address (Customer Entity).
- Add new customer if their phone number cannot be found
- Show total sales by tracking payments received and the type of the payments (cash or card) (Customer & Payment Entities), as well as keeping track of the cash in the drawer (Type Attribute in Payment Entity).
- Select items to be added to an order and display final price at the end (Product Entity).
- Print a receipt of the order along with information such as the customer name, id, address(if applicable), product total paid, & total cost (Order Entity),
- Update stock quantity when item is ordered (Stock Entity).
- Add and remove products from an order
- Combining and splitting orders for payment purposes

Phase 2: Logical and Physical Database Design

Relation Set



Customer_Order? -1

Attribut description table? -3

For professionalism, don't allow content to creep into the margins.

Example Data Set

Order

<u>ID</u>	Total	Tax	Order_Cost	Time	Still_Owed
1	\$27.80	1.07	\$26.73	10:20:28 2017-10-26	\$0.01

Customer

<u>Phone_number</u>	ID	Name	Amount_Paid	Time
555-573-1234	5555731234	John Smith	\$27.79	10:20:28 2017-10-26

Payment

<u>Order_ID</u>	Cust_ID	Type	Amount	Left_In_Drawer	Card_Number
5555731234-1	5555731234	Card	\$27.79	\$125	1234567894946251

Product

<u>ID</u>	Name	Type	Base_cost	Base_Usage	Multiplier	Total
1	Pizza	Pep	\$7.99	2	1	\$8.99

Stock

<u>ID</u>	Item	Type	Available_Quantity
41	Dough	Flat Crust	4
21	Topping	Pepperoni	5

Order_Products

order_id	product_id	Size	Additions	Add_Cost
5555731234-1	1	L	Ex. Cheese	\$1.00

Product_stock

Product_ID	Stock_id	Base_Usage
1	1	5


Application Program Design

Function (1): Identify customer by phone number

//This function is to identify the Customer calling to make a order. It access the *Customer* table

Input: Customer Phone Number

Steps:

1. Search the Customer table for the customer by phone number
2. If a customer is found, display customer information and proceed to create order page with customer information loaded.
3. If customer is not found, go to create customer page 

Function (2): Create an Order

//This function is used to create an order. It accesses the Order table. The saved record will be updated when the order is confirmed

Input: Customer ID, Time & Date Timestamp

Steps:

1. Display the customer information from the *Customer* table using the customer ID provided
2. Display the date using the timestamp provided.
3. Save the Order.

Function (3): Add an Item

Creates into Margin

read Customer table

Page break

//This function adds an item to the order. It accesses the Order_Products table, the Order table & the Product table.

Input: Product Id, Order Id, Size

Steps:

1. Insert a new record to the *Order_Product* table ^{with $-\frac{1}{2}$} the product_id , the order_id and the size.
2. Get the base cost of the product from the *Order* table and multiply it with the corresponding multiplier (2, 1.5 or 1) depending on the size of the product.
3. Update the order (from the *Order* table) by adding the cost from step 2 to the order_cost attribute.
4. Update the order page to show the new product added.

Function (4): Remove an Item

//This function removes a product from an order. It accesses the Order_Products table.

Input: (\$product_id, \$order_id)

Steps:

1. Delete from Order_Products Where Order_Id = \$order_id and Product_Id = \$product_id
2. Update the Order page to show that a item has been deleted from the order.

Function (5): Confirm an Order

//This product confirms an order, calculates the tax and creates a new payment if the payment is by card. It accesses the Order table and the Payments Table, the Customer table, the Product_Stock table & the Stock table.

Input: Order Id

Steps:

1. Find the Order by the Order Id.
2. Update the tax attribute by multiplying the tax rate (2.6%) from the order_cost.
3. Save the record.
4. Show confirmation message.
5. Ask if the payment is by card or cash.
6. If the type is card, create a payment where the type is card, the amount is the total of the tax & order_cost selected from the *Order* table.
7. If the type is cash, a new payments record is created once the cash is received.
8. The customer's amount paid attribute is updated by adding the total cost of the current order to it.
9. For each product:
 - a. For each ingredient in the product:
 - i. Find how much of the ingredient is used per product from the Product_Stock table.
 - ii. Multiply this with the corresponding multiplier (2, 1.5 or 1) depending on the size of the product.

- iii. Subtract this value from Available_Quantity in the Stock table for the current ingredient in the loop.

Function (6): Create a Payment

//This is used to create a new payment when the cash is physically received. It accesses the Payments table & Customer table.

Input: \$order_id

Steps:

1. The \$order_id is used to find the \$customer_id.
2. A new record is created in the payments table where order_id = \$order_id & customer_id = \$customer_id.
3. The type is selected is cash, and the amount is the money provided.
4. The payment record is saved.
5. The customer's amount paid attribute is updated by adding the total cost of the current order to it.

Function (7): Cancel an Order.



//This deletes the order and all relevant records in the Order_Products table. It accesses the Order table and the Order_Products table.

Input: (\$order_id)

Steps:

1. Delete from Order_Products Where Order_Id = \$order_id
2. Delete from Order Where id=\$order_id
3. Show cancellation message.

Function (8): Split an Order

//This takes one order and splits it to 2 or more orders. It uses the Payment table, the Order table & the Customer table.

Input: \$order_id, number of orders to split the order to.

Steps:

1. Find the Order by the order id from the Order table.
2. For the number of orders to split the order to:
 - a. Call function 2 (Create an Order)
 - b. The user selects from the above order, the items that need to be added to this split order.
 - c. For each item selected:
 - i. Call function 3 (Add an item)
3. Call function 7 (Cancel an Order) by passing in \$order_id.

Function (9): Print an Order

page break

//This simply transitions to a page which only shows the order. It uses the Order table and the Order_Products table.

Input: \$Order_Id

Steps:

1. Transition to the /orders/:order_id page.
2. Load data about the order by using order id and update the page.
3. Find all the items by searching the Order_Products table where order_id = \$Order_Id
4. For each item, update the order page by adding the item as a row in a html table.
5. The user prints by using the web browser to print the page.

User Interface

Order #25			
#	Name	Size	Cost
1	Pepperoni	Small	10.00
2	Sausage	Large	16.00
3	Coca Cola	Medium	5.00
4	Bread Sticks	Small	6.99

Devinda Senanayake
573-212-5682

Order Total: \$ 37.99
Tax: \$ 0.98
Total: \$ 38.97

Order page (orders/:order_id) ready to be printed

Order #25

#	Name	Size	Cost	
1	Pepperoni	Small	10.00	✕
2	Sausage	Large	16.00	✕
3	Coca Cola	Medium	5.00	✕
4	Bread Sticks	Small	6.99	✕



Devinda Senanayake
573-212-5682

Order Total: \$ 37.99

Tax: \$ 0.00

Total: \$ 0.00

Confirm

Print

Cancel

Create an Order Page

Adding to Order #25

Select an Item ▼

Pepperoni
Cheese Pizza
Bread Sticks
Sausage Pizza

Select a Size

- ☒ Small
☐ Medium
☐ Large

Add

Cancel

Add an Item to an Order. (Opens as a Modal)