Anders Quigg ahq6qa

Hong Moon hsm5xw

Daniel Katona djk5as

CS4750 Semester Project

Introduction:

Our project is to create the database for an online sporting goods store. This database will be used by customers to place and manage orders and payment information, and by the store to manage inventory and distribution. Each customer has a unique account, which stores identification, payment, and delivery information. The various products that the store has to offer are kept in the database for display, search, and selection in the front-end website, and the details of the vendors providing those products and the shipments and amounts received from each are also kept to help the store manage inventory. Customers can also leave product reviews for the benefit of other customers.

Requirements:

Customers should be able to create, sign into, and log out of accounts. Accounts need to store personal details such as names, addresses, and contact information, unique account identifiers and passwords, as well as a record of past orders for each customer. Customers can also store payment and credit card information to expedite purchases. The details of each product that the store has to

offer are also stored in the database; these details include unique identifiers, product names, prices, descriptions and pictures, as well as inventory information such as amount left in stock. While making orders, customers build shopping carts, which store the amount of each product the customer desires to purchase at that time. When the cart has been completed, the order is placed, along with the date of the order, the total price, payment details, and the order's delivery status. The vendor of each product is also maintained in the DB, along with contact information. Reviews consist of a message along with a 1-5 star rating.

Security is handled with safeguards against SQL injection attacks, as well as mysql user types. Unknown users default to a "login" user, which is limited to selecting from the customer table to determine a username-password match. After that check is passed, users are transferred to a "customer" user, which is limited to performing the necessary operations on customer-related tables within the database. Special functionality is provided by a number of triggers and stored procedures, to be discussed below.

Table Details:

Customer

- To register as a Customer, one must insert customer_id (as username), password, name, address, phone number.
- A Customer is identified with customer_id.
- There should be no customers with the same customer_id.
- A password should consist of alpha-numeric characters, and its length should be at least 8.
- We break down the customer's name into first and last name
- We assume that each customer has only one address, so we do not treat address as multi-valued.
- An address should include street address, city, state, and 5-digit zip code.
- Since an address is a composite attribute, we made a decision to include street address, city, state, and zip code instead of one address.

- A phone numbers should have 10 digits in the U.S.
- We assume that we only request one phone number for a customer, so we do NOT treat phone number as multi-valued.

Product

- A Product has a unique id number.
- A Product also has product name, amount left in stock, price, vendor name, date of the product supplied, and product description.
- We make an assumption that vendor name is a unique identifier for a vendor.
- A price for each product must be greater than 0 dollar.
- We also store the URL to which the image of the product will be provided by an external hosting website

Orders

- An Order has a unique order id number for each order.
- A Customer can order many Products, and one Product can be ordered by many Customers.
- When a Customer Orders a Product, the following information should be recorded: order id, customer id, a product list id (which keeps track of the list of products purchased with the amounts), order date, total order price, order status, and payment_method_id.
- An order status is one of the following: not shipped yet, in-transit, delivered.
- An order has a payment_method_id, which is the credit card billing information from the Customer.
- When a Customer makes an Order, the order information should be stored in Order_History for that Customer.
- A Customer can only order a Product if the amount left in stock is greater than or equal to the order amount of the product. If there is not enough product left in stock, the Customer should receive a message that the order can't be done because there is not enough product left in stock.
- We made a decision to keep the total_order_price in the Order table, because it doesn't make sense to put the price in the Payment_Method table, which is to primarily store credit card information.

Product_list

- A Product_list includes information about the list of products with the amounts that a customer has chosen for an order.
- We made a decision to make this Product_list table so that we keep track of the list of products a Customer intends to purchase. This decision will enable the Customer to buy more than one different products per Order.
- A Product_list is identified with list_id and product_id.
- A Product_list should have list_id, product_id, and the amount of the product to order.

Order_history

- When a Customer makes an Order, the order information should be stored in Order History for that Customer.
- An Order_history should have a customer id and an order id.

Vendor

- A vendor supplies a particular product
- A Vendor is identified by vendor name.
- We make an assumption that vendor name is a unique identifier for a vendor.
- A Vendor should have contact information: vendor name, phone number, address, email address of contact personnel.
- An address should street address, city, state, and 5-digit zip code.
- A phone number should have 10 digits in the U.S.
- We assume that we only request one phone number from a vendor, so we do NOT treat phone number as multi-valued.

Supply

- Each Product is supplied by one Vendor, but one Vendor can supply many different Products.
- An entry in the Supply table should include a product id and a vendor name.
- When a Vendor supplies a product, the following information should also be recorded: supplied date, supplied amount.

Message

- A customer sends private Messages to the website for question, complaints, and follow-ups, such as “I received a wrong product.”
- A Message has message id, customer id, message title, message content, date
- A Message is identified by message id.

Product_review

- A Product_review is identified with product_review_id
- A Product_review should have a product_id, customer_id, date, star_rating.
- A star_rating must be an integer in the range of 1 to 5.

Payment_method

- A Payment_method table stores credit card payment information for the order purchase.
- To simplify the payment methods, we only allow payments with credit cards
- A Payment_method table is identified with payment_method_id
- A Payment_method table should have payment_method_id, customer_id, credit_card_num, credit_card_type, and csv number.
- A credit card number should consist of 16 digits.
- A credit card type refers to the type of the credit card, such as Visa or Master Card.
- A csv number should consist of 3 digits.

Design Process:

We chose a website as the interface application for ease of connection with customers. For the purposes of our project, security was handled by data/code separation to guard against SQL injection and the management of user types within the database; no data is encrypted, though this might be a useful feature to add in the future.

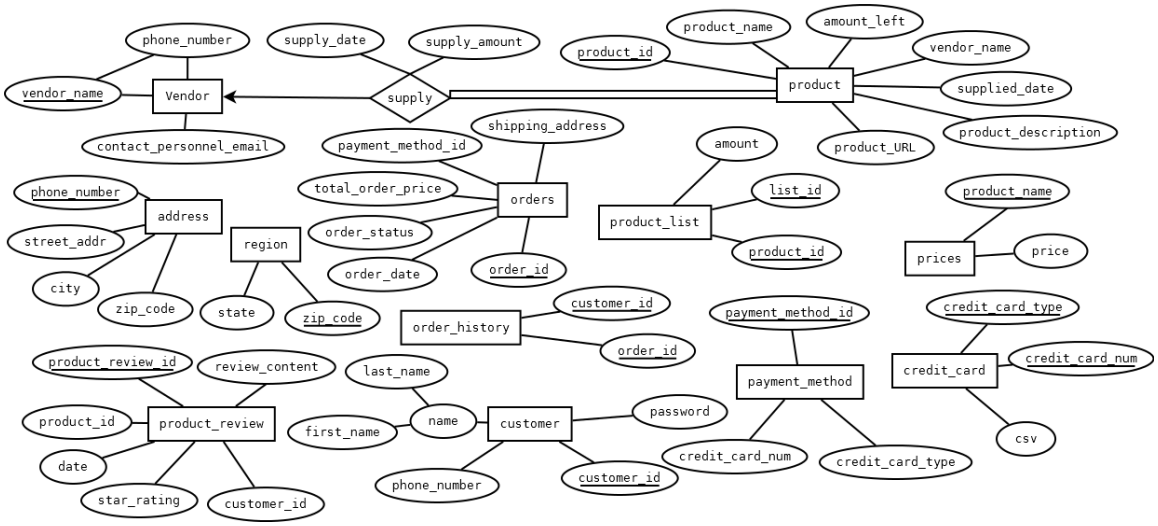
The database is designed around three principal areas; customer information, purchase and product information, and store inventory and supply information. The customer information tables hold the contact and delivery information of customers, as well as the account details that let them interact with the database and the site. Purchase and product tables allow customers to browse and place orders for products that the store offers, and allows the store to manage and keep records of these transactions. The inventory and supply tables let the store keep track of vendors and suppliers, as well as products and their amounts and dates in inventory.

Triggers and stored procedures and functions are used in the database to ease interfacing. One function checks to see if the products in stock are sufficient to meet an order that a customer is attempting to place. Another calculates the total price of an order, while another returns the next available product list ID. Auto-increment was not an option for this last example, as multiple products can exist within the same order, requiring multiple instances of the same ID. Additionally, a trigger detects inserts in the order table (representing the customer placing an

order), and automatically adds the order details to the customer's user history and uses a stored procedure to decrement the amount of each product remaining by the appropriate quantity. Unfortunately, problems in developing php code prevented the use of the functions that return a value; however, they are fully functional within the database and can be seen and tested in phpMyAdmin. Instead, the next available list_id is obtained with php code, and the product in stock check is unimplemented in the front-end application. The trigger and its associated procedure are working and fully integrated with the front-end application.

It is assumed that the administrative users of the database will be able to interact from the phpMyAdmin back-end. As such there is no admin login or functionality in the front end, and administrative transactions such as adding new products and vendors, deleting customer information, and exporting to XML are performed in phpMyAdmin.

ER Diagram:



Database Schema:

address:

```
`phone_number` varchar(12) NOT NULL DEFAULT '',
 `street_addr` varchar(40) NOT NULL,
 `city` varchar(25) NOT NULL,
 `zip_code` varchar(5) NOT NULL,
 PRIMARY KEY (`phone_number`)
```

credit_card:

```
`credit_card_num` varchar(16) NOT NULL DEFAULT '',
 `credit_card_type` varchar(20) NOT NULL DEFAULT '',
 `csv` varchar(3) NOT NULL,
 PRIMARY KEY (`credit_card_num`,`credit_card_type`)
```

customer:

```
`customer_id` varchar(20) NOT NULL DEFAULT '',
 `password` varchar(20) NOT NULL,
 `first_name` varchar(20) NOT NULL,
 `last_name` varchar(20) NOT NULL,
 `phone_number` varchar(12) NOT NULL,
 PRIMARY KEY (`customer_id`),
 UNIQUE KEY `customer_id` (`customer_id`)
```

orders:

```
`order_id` int(11) NOT NULL AUTO_INCREMENT,
 `customer_id` varchar(20) NOT NULL,
```

`list_id` varchar(8) NOT NULL,

`order_date` varchar(20) NOT NULL,

`total_order_price` float NOT NULL,

`order_status` varchar(180) NOT NULL,

`payment_method_id` varchar(8) NOT NULL,

PRIMARY KEY (`order_id`),

FOREIGN KEY (`order_id`) REFERENCES `orders` (`order_id`)

FOREIGN KEY (`customer_id`) REFERENCES `customer` (`customer_id`)

order_history:

`customer_id` varchar(20) NOT NULL DEFAULT '',

`order_id` int(11) NOT NULL,

PRIMARY KEY (`customer_id`,`order_id`),

KEY `order_id` (`order_id`)

payment_method:

`payment_method_id` int(11) NOT NULL AUTO_INCREMENT,

`credit_card_num` varchar(16) NOT NULL,

`credit_card_type` varchar(20) NOT NULL,

PRIMARY KEY (`payment_method_id`)

prices:

`product_name` varchar(50) NOT NULL DEFAULT '',

`price` float NOT NULL,

PRIMARY KEY (`product_name`)

product:

`product_id` varchar(8) NOT NULL DEFAULT '',

`product_name` varchar(50) NOT NULL,

`amount_left` int(11) NOT NULL,

`vendor_name` varchar(20) NOT NULL,

`supplied_date` varchar(20) NOT NULL,

`product_description` varchar(180) NOT NULL,

`product_url` varchar(200) NOT NULL,

PRIMARY KEY (`product_id`),

FOREIGN KEY (`vendor_name`) REFERENCES `vendor` (`vendor_name`)

product_list:

`list_id` varchar(8) NOT NULL DEFAULT '',

`product_id` varchar(8) NOT NULL DEFAULT '',

`amount` int(11) NOT NULL,

PRIMARY KEY (`list_id`,`product_id`)

product_review:

`product_review_id` varchar(8) NOT NULL DEFAULT '',

`product_id` varchar(8) NOT NULL,

`customer_id` varchar(20) NOT NULL,

`date` varchar(20) NOT NULL,

`review_content` varchar(450) NOT NULL,

`star_rating` int(11) NOT NULL,

PRIMARY KEY (`product_review_id`),

FOREIGN KEY (`product_id`) REFERENCES `product` (`product_id`)

FOREIGN KEY (`customer_id`) REFERENCES `customer` (`customer_id`)

region:

`zip_code` varchar(5) NOT NULL DEFAULT '',

`state` varchar(2) NOT NULL,

PRIMARY KEY (`zip_code`)

supply:

`vendor_name` varchar(20) NOT NULL DEFAULT '',

`product_id` varchar(8) NOT NULL DEFAULT '',

`supplied_date` varchar(20) NOT NULL,

`contact_personnel_email` varchar(45) NOT NULL,

`supplied_amount` int(11) NOT NULL,

PRIMARY KEY (`vendor_name`,`product_id`),

FOREIGN KEY (`vendor_name`) REFERENCES `vendor` (`vendor_name`)

FOREIGN KEY (`product_id`) REFERENCES `product` (`product_id`)

vendor:

`vendor_name` varchar(20) NOT NULL DEFAULT '',

`phone_number` varchar(12) DEFAULT NULL,

`contact_personnel_email` varchar(45) NOT NULL,

PRIMARY KEY (`vendor_name`)

Product Evaluation:

Testing procedures principally consisted of the comparison of the expected and actual values of operations and queries. Each query in the front end application was run individually, followed by the examination of the database in phpMyAdmin to confirm that the correct operation had taken place or the correct table was returned. The stored procedures and triggers within the database were tested in a similar fashion. Some sample queries and results are below:

A: Find the next available product list_id to build a new order

- SELECT getNextListId() → 5

B: Find the total price of an order with product_list_id of 3

(Tennis racquets 1xE3 and 3xD6)

- SELECT getTotalPrice(3) → 286.78

C: Display the orders made by John Doe (Customer03)

- SELECT * FROM orders WHERE customer_id = 'Customer03'

→ (order_id, customer_id, list_id, order_date, total_order_price, order_status, payment_method_id)

=

3	Customer03	3	2013-11-01	286.78	Shipped	3

Third Normal Form proofs:

(1) Customer

Calculate F+:

- **1. Functional Dependencies:**

  customer_id→ customer_id, password, First name, Last name, street_addr, city, state, zip_code, phone_number

  phone_number→ street_addr, city, state, zip_code

  zip_code→ state

- **2. Add Reflexives**

  customer_id→customer_id, password, First name, Last name, street_addr, city, state, zip_code, phone_number

  phone_number→phone_number, street_addr, city, state, zip_code

  zip_code→zip_code, state

- **3. No Additional Rules apply so this is F+**

Calculate Fc:

- **1. Remove Reflexives:**

  customer_id→password, First name, Last name, street_addr, city, state, zip_code, phone_number

  phone_number→street_addr, city, state, zip_code

  zip_code→ state

- **2. Eliminate Transitive Dependencies:**

  customer_id→phone_number

phone_number→street_addr, city, state, zip_code

customer_id→street_addr, city, state, zip_code

Eliminate

customer_id→street_addr, city, state, zip_code

To Get:

customer_id→password, First name, Last name,phone_number

phone_number→street_addr, city, state, zip_code

zip_code→ state

- **3. Eliminate Transitive Dependencies:**

  phone_number→zip_code

  zip_code→ state

  phone_number→ state

  Eliminate:

  phone_number→ state

  This yields the final Fc of:

  customer_id→password, First name, Last name, phone_number
  phone_number→street_addr, city, zip_code
  zip_code→ state

Result:

The following 3NF tables are now taken directly from Fc:

customer_id, password, First name, Last name, phone_number //

phone_number, street_addr, city, zip_code //

zip_code, state

These are will be called *customer*, *address*, and *region*, respectively, in our database.

(2) Product

Calculate F+:

- **1. Functional Dependencies:**

  product_id→ product_id, product_name, amount_left, price, vendor_name, supplied_date, product_description, URL

  product_name→ price

- **2. Add Reflexives**

  product_id→ product_id, product_name, amount_left, price, vendor_name, supplied_date, product_description, URL

  product_name→ product_name, price

- **3. No Additional Rules apply so this is F+**

Calculate Fc:

- **1. Remove Reflexives:**

  product_id→product_name, amount_left, price, vendor_name,
  supplied_date, product_description, URL

  product_name→price

- **2. Eliminate Transitive Dependencies:**

  product_id→product_name

  product_name→price

  product_id→ price

  Eliminate
  product_id→ price

  To Get:

  product_id→product_name, amount_left, vendor_name,supplied_date,
  product_description, URL

  product_name→price

- **3. Since there are no more transitive dependencies, this is the final Fc.**

Result:

The following 3NF tables are now taken directly from Fc:

product_id, product_name, amount_left, vendor_name,supplied_date,
product_description, URL //
product_name, price

These will be called *product* and *prices*, respectively, in our database.

(3) **Order**:

- **Functional Dependencies:**

  order_id→order_id, customer_id, list_id, order_date, total_order_price, order_status, payment_method_id

- **Because there is only one functional dependency and order_id is the super key of this table, this table is already in 3NF.**

- **This yields the following table:**

  order_id, customer_id, list_id, order_date, total_order_price, order_status, payment_method_id

This table will be called *orders* in our database.

(4) **Product list**:

- **Functional Dependencies:**

  list_id, product_id→list_id, product_id, amount

- **Because there is only one functional dependency and the combination of list_id and product_id is the super key of this table, this table is already in 3NF. This yields the following table:**

  list_id, product_id, amount

This table will be called *product_list* in our database.

(5) **Order history**:

- **Functional Dependencies:**

  customer_id, order_id→customer_id, order_id

- **Because there is only one functional dependency and the combination of customer_id and order_id is the super key of this table, this table is already in 3NF. This yields the following table:**

  customer_id, order_id

This table will be called *order_history* in our database.

(6) **Vendor**:

Calculate F+

- 1. **Functional Dependencies:**

  vendor_name→vendor_name, phone_number, address, contact_personnel_email

  phone_number→ address

  (Note: As explained in the Requirements document, we treat the vendor address as atomic so we don't break it apart as in Customer table)

- **2. Add Reflexives**

  vendor_name→vendor_name, phone_number, address, contact_personnel_email

  phone_number→phone_number, address

- **3. No Additional Rules apply so this is F+**

Calculate Fc:

- **1. Remove Reflexives:**

  vendor_name→phone_number, address, contact_personnel_email

  phone_number→address

- **2. Eliminate Transitive Dependencies:**

  vendor_name→phone_number

  phone_number→ address

  vendor_name→ address

  Eliminate

  vendor_name→ address

To Get:

vendor_name→phone_number, contact_personnel_email

phone_number→address

This yields the final Fc of:

vendor_name→phone_number, contact_personnel_email

phone_number→address

Result:

The following 3NF tables are now taken directly from Fc:

vendor_name, phone_number, contact_personnel_email //

phone_number, address

The first table will be called *vendor* in our database and since the second table already has the same information contained in the address table created when we did customer, we will not add a new table for phone_number and address.

(7) **Supply**:

- **Functional Dependencies:**

product_id, vendor_name → product_id, vendor_name, supplied_date, supplied_amount

- **Because there is only one functional dependency and the combination of product_id and vendor_name is the super key of this table, this table is already in 3NF. This yields the following table:**

product_id, vendor_name, supplied_date, supplied_amount

This table will be called *supply* in our database.

(8) **Product_review**:

- **Functional Dependencies:**

  product_review_id→ product_review_id, product_id, customer_id, date, star_rating

- **Because there is only one functional dependency and product_review_id is the super key of this table, this table is already in 3NF. This yields the following table:**

  product_review_id, product_id, customer_id, date, star_rating

This table will be called *product_review* in our database.

(9) **Payment_method**:

Calculate F+:

- **Functional Dependencies:**

  payment_method_id→ payment_method_id, customer_id, credit_card_num, credit_card_type, csv

  credit_card_num, credit_card_type→ csv

- **Add Reflexives**

  payment_method_id→payment_method_id, customer_id, credit_card_num, credit_card_type, csv

  credit_card_num, credit_card_type→credit_card_num, credit_card_type , csv

- **No Additional Rules apply so this is F+**

Calculate Fc:

Remove Reflexives:

payment_method_id→ customer_id, credit_card_num, credit_card_type, csv

credit_card_num, credit_card_type→csv

Eliminate Transitive Dependencies:

payment_method_id→credit_card_num, credit_card_type, csv

credit_card_num, credit_card_type→csv

payment_method_id→csv

Eliminate

payment_method_id→csv

To Get:

payment_method_id→credit_card_num, credit_card_type

credit_card_num, credit_card_type→csv

This yields the final Fc of:

payment_method_id→credit_card_num, credit_card_type

credit_card_num, credit_card_type→csv

The payment_method tables are now taken directly from Fc to yield the following in 3NF:

payment_method_id, credit_card_num, credit_card_type //

credit_card_num, credit_card_type, csv

These are will be called *payment_method* and *credit_card* respectively, in our database.