Open your mind. LUT.
Lappeenranta University of Technology

CT60A4304

Basics of database systems
Iflaah Salman

ONLINE STORE MANAGEMENT SYSTEM

THEORETICAL REPORT

Group 22

Members: Noora Alikirri, Thet Htar Zin

Table of contents

1. Introduction

Imaginary Business Background

In pursuit of practical work experience and financial gains, we have embarked on the journey of launching a small business. Utilizing the knowledge and skills acquired throughout our academic journey, we are establishing an online shop where we sell fashionable jewelries. To effectively manage our customer and product data, the necessity for a database management system has been brought up. Additionally, we intend to develop a simple, user-friendly application integrated with the database that will allow us to manage and maintain our database.

Potential Challenges

As a two-person team venturing into this business, there are some challenges we could face with this business model posed by limited financial and human resources so in the beginning, our capacity to build a highly robust, sophisticated database management system is restrained. Thus, we aim to initiate with something small and manageable. Acknowledging our limited IT expertise, we can face challenges in terms of solving technical issues. During the first year after the opening of the online store, we aim for 5-15% profit and a reasonable percentage of the customer base so we might not need to tackle scalability issues.

Motivation for Choosing Domains and Their Descriptions

For our online store management system, we'll have six entities which are product, warehouse, customer, order, supplier and productorder, that joins information from order and product entities.

PRODUCT

The purpose of the table of 'product' is to store the data of each product we're selling. We'll have unique identifier of an item which is its ID, item name, and price. We also have warehouseID as foreign key, so that we know in which warehouse the product is being stored and the number of products stored in the warehouse. Item ID is a primary key and must not be null, and it's a string or characters. The name of the product is a string or characters. The price of the product is also a positive integer and should not be null. Otherwise, it would be like offering products for free and would result in a profit loss. The amount of product can be zero or a positive integer, since it's possible that we don't have the product currently available.

WAREHOUSE

The entity 'warehouse' is to know the location of the warehouse that stores the products available. As attributes, we will have unique identifier of a warehouse which is the primary key, and the city where the warehouse is located. Warehouse ID must be positive integer and must not be null. The city can be a string or characters.

CUSTOMER

The entity of 'customer' is to store the data of customers. Here, we'll have attributes such as the ID of each customer which is the primary key, the name of the customer and their address so that when they make a purchase, we'll know who they are, and which address the product should be delivered to. The ID of the customer must be positive integer and must not be null. The name of the customer and address can be string or characters and must not be null either.

ORDER

With this 'order' entity, we can know what kind of product the customer ordered and who ordered. The attributes are order ID, which is the primary key, customer ID which is the foreign key referenced from customer entity. The date and time of the order must be in datetime format.

SUPPLIER

Since we are not the producers of the products but simply an online store selling those products, we need a table to store supplier information. For this purpose, we'll have a supplier entity with attributes such as supplier ID as a primary key, supplier's name and contact number. We also have productid as a foreign key, so that we know what product that supplier is producing. The ID of the supplier must be positive integer and must not be null. The name of the supplier can be stored as string or characters. Supplier contact numbers must be in positive integer. The productID must not be null and can be stored as string or characters.

PRODUCTORDER

The entity productorder holds information about the products in a specific order. It has orderID and productID as foreign keys and the amount of each product ordered, so that we'll know how much to bill from a customer. OrderID and amount of product are positive integers and productID is a string or characters.

Business rules of each entity

PRODUCT

Each product ID must be different from each other. The price of the product must not be zero or less than zero. It must be always greater than zero. The number of products must be either zero or greater than zero and must not be stored in a negative integer.

WAREHOUSE

Each warehouse must have a different ID and must have the product's ID and amount of it. The address of the warehouse must have detailed street names and postcode. Each warehouse can have many products, but each product can be stored only in one warehouse which means warehouse entity has one-to-many relationship with product entity.

CUSTOMER

The customer ID must be different for every customer and the name of the customer must be stored as first name and last name. The shipping address or the phone number of the customer must not be null. The phone number must have country code in it.

ORDER

The order must be associated with a specific customer so each order must include order ID and customer ID. The order must also include the date when the customer made the order. Each order can include one or many products. A customer can make multiple orders, but each order is related to only one customer, so there is a one-to-many relationship between order and customer.

SUPPLIER

Each supplier must have a unique and different ID. Each supplier can provide one or many products, but a product can be supplied only from one supplier so there's a one-to-many relationship between

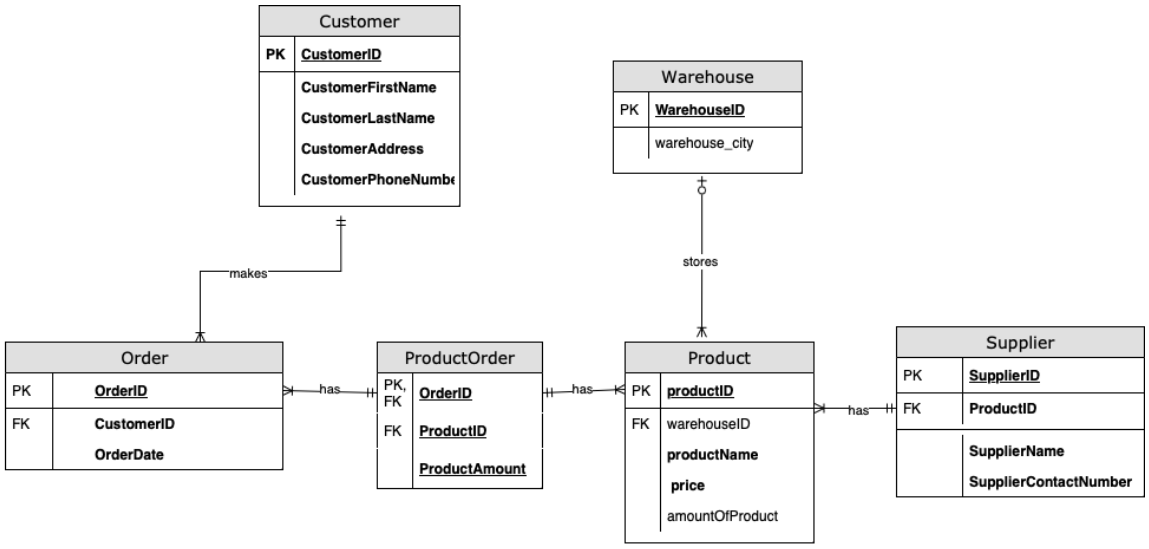
supplier and product. Contact information of each supplier must be different. The supplier's name must be provided.

PRODUCTORDER

Productorder entity is used to solve the many-to-many relationship between order and product entities. The products in an order must be associated with a specific customer. The order must also include the total amount for each product ordered.

2. ERD of the real-world problem

For our business model, we chose Crow's foot notation to design ERD.



3. Relational model transformed from ERD

Product (ProductID (PK), WarehouseID (FK), ProductName, Price, AmountOfProduct)

Order (OrderID (PK), CustomerID (FK), ProductAmount, OrderDate)

ProductOrder (ProductID (FK), OrderID (PK,FK), ProductAmount)

Warehouse (WarehouseID (PK), Warehouse_city)

Customer (CustomerID (PK), CustomerFirstName, CustomerLastName, CustomerPhoneNumber CustomerStreetAdress, CustomerCityAddress)

Supplier (SupplierID (PK), ProductID (FK), SupplierName, SupplierContactNumber)

4. Improvements applied to the DB structure

We noticed that in Customer entity we had address, which is a composite attribute, so we broke it down to more simple attributes: Street and City. We also had one many-to-many relationship between Order and Product entities, so we had to add a new entity between them, so that one order can have multiple products and products could belong to multiple orders. In the end we didn't have to use that many improvements compared to our initial database structure, since we tried to keep it quite simple and manageable.

Indices are useful where query statements such as WHERE, JOIN and aggregations are used multiple times. By creating indices, we can improve the performance of the queries and reduce the running time of our application. So, for our online store, we notice that viewing order details include execution of LEFT JOIN so if we want to view it multiple times, we will need an index. Thus, we decided to create an index for order date so that we can filter out the orders based on date.

We implemented the following query.

```
CREATE INDEX idx_order_date ON Order (OrderDate);
```

5. Brief description of the application

We're developing a simple application where we can manage our database via a simple GUI. The following are the functions that can be executed with our program.

1) Through the main menu user can choose the desired topic: managing customers, products, orders, warehouses or suppliers. Inside each of these topics there are all the implemented functions for that specific topic.

2) Inserting new customer information, update or delete the existing ones.

3) Insert new products, update or delete the existing ones.

4) Insert, update or delete the data of suppliers of our products.

5) Insert, update or delete the data of warehouses.

6) Take record of the order details by inserting or deleting. There is also a possibility to search for an order.

7) View all the existing products

8) View all the existing orders

9) View all the existing customers

10) View all the existing warehouses

11) View all the existing suppliers

12) View all the existing orders ordered by date

To develop this simple program which doesn't require fast execution time since our business is still new, we decided to build it with python programming language.

A list of the SQL queries implemented in the program.

1) **To add new customer to our database,**

INSERT INTO Customer (CustomerID,CustomerFirstName,CustomerLastName, CustomerAddress, CustomerPhoneNumber)

VALUES (%s, %s, %s, %s, %s)

2) **To delete the customer details using their ID,**

DELETE FROM Customer WHERE CustomerID=%s

3) **To show the details for existing customers**

SEARCH * FROM customer

4) **To insert new product information to the database,**

INSERT INTO product (productID, productName, price, warehouseID)

VALUES (%s, %s, %s, %s)

5) **To view the updated details of a product,**

SELECT * FROM product

6) **To check if a product exists by using product ID,**

SELECT productID FROM product WHERE productID = (%s)

7) **Updating product details,**

UPDATE product SET productName = (%s), price = (%s), warehouseID = (%s), amountofproduct = (%s) WHERE productID = (%s)

8) **Deleting a product,**

DELETE FROM product WHERE productID = (%s)

9) **To view all orders based on orderID (combining information form tables order and productorder:**

SELECT order.orderID, order.customerID, order.orderDate, productorder.productID, productorder.productAmount FROM order LEFT JOIN productorder ON order.orderID = productorder.orderID

10) **To add a new order:**

INSERT INTO order (orderID, customerID, orderdate)
VALUES (%s, %s, %s);
INSERT INTO productorder(orderID, productid, productAmount)
VALUES(%s, %s, %s, %s);

11) **To view or search for an order:**

SELECT orderID, customerID, orderDate FROM order WHERE orderID=(%s)
SELECT orderID, productID, productAmount FROM productorder WHERE orderID =(%s)

12) **To delete an order:**

SELECT orderID FROM order WHERE orderID = (%s)
DELETE FROM productorder WHERE orderID = (%s)
DELETE FROM order WHERE orderID = (%s)

13) **To add a new Warehouse:**

INSERT INTO warehouse (WarehouseID, warehouse_city) VALUES (%s, %s)

14) **To update Warehouse location:**

SELECT WarehouseID FROM warehouse WHERE WarehouseID = (%s)
UPDATE warehouse SET warehouse_city = (%s) WHERE WarehouseID = (%s)

15) **To delete a Warehouse:**

SELECT WarehouseID FROM warehouse WHERE WarehouseID= (%s)
DELETE FROM warehouse WHERE WarehouseID = (%s)

16) **To add a new Supplier:**

INSERT INTO supplier (SupplierID, SupplierName, ProductID, SupplierContactNumber) VALUES (%s, %s, %s, %s)

17) **To update Supplier information**

UPDATE supplier SET SupplierName = (%s), ProductID = (%s), SupplierContactNumber= (%s) WHERE SupplierID = (%s)"

18) **To delete Supplier:**

SELECT SupplierID FROM supplier WHERE SupplierID= (%s)
DELETE FROM supplier WHERE SupplierID = (%s)

19) **To view all the existing orders ordered by date:**

SELECT order.orderID, order.customerID, order.orderDate, productorder.productID, productorder.productAmount FROM order LEFT JOIN productorder ON order.orderID = productorder.orderID ORDER BY order.orderDate

Imagined Scenarios per Chosen Domain and Its Business Rules

Customer Information

1) **Imagined scenario:** A new customer signed up or registered to buy our products online. So, we need to add them to the database.
   Implementation of business rules: Use INSERT statement to add details of the new customer making sure the ID, their first name and last name, address and phone number are not null.

2) **Imagined scenario:** We accidentally wrote the last name of the customer wrong, so we need to update it based on their ID.
   Implementation of business rules: Use UPDATE statement to edit the details of the customer making sure all information is filled in correctly.

3) **Imagined scenario:** A customer no longer wants to buy products from us (hopefully not), and we need to remove them from our database.
   Implementation of business rules: Use DELETE statement to remove all details of the customer.

4) **Imagined scenario:** We decided to give a discount to our customers but only if they're customers before buying the product, so we need to check if their data is stored in our database.
   Implementation of business rules: Use SELECT statement with the condition WHERE to find the customer's details given that we know their ID.

Product Management

1) **Imagined scenario:** We decided to sell a new product in our online store, so we need to insert new details of the product, the amount of it and which supplier we're getting the product from.
   Implementation of business rules: Use INSERT statement to add details of the product and make sure that product ID, name and warehouse ID are not null. The price must be greater than zero.

2) **Imagined scenario:** We found out a product is not stored in the warehouse with specific ID but the other one, so we need to change it.
   Implementation of business rules: Use UPDATE statement to make changes to the correct warehouse ID where the product is stored. Make sure to include all the details when updating the information.

3) **Imagined scenario:** We no longer sell a product because we no longer have a contract with a supplier.
   Implementation of business rules: Use DELETE statement to remove the details of the product that is no longer available.
4) **Imagined scenario**: A customer is interested in our product and would like to buy 20 of them if they're still available in our warehouse but we do not know how many of them are left in the warehouse, so we need to check it.
   Implementation of business rules: Use SELECT statement with WHERE condition to search for a specific product. We can use product ID for the condition statement.

Warehouse Inventory

1) **Imagined scenario:** We have newly imported products that we must store in our warehouse, so we want to search for the location of the warehouse.
   Implementation of business rules: Use SELECT statement to print out the details of our warehouses.

2) **Imagined scenario:** We have more products now than ever so our two warehouses cannot store them all and so we have recently rented a warehouse. We want to add the details to our database.
Implementation of business rules: Use INSERT statement to add details of our new warehouse making sure that both the ID and location are not null.

3) **Imagined scenario:** We noticed that renting the warehouse costs a lot, so we decided to stop renting it and remove it from our database.
Implementation of business rules: Use DELETE statement to remove the details of the warehouse.

Order Placement

1) **Imagined scenario:** A customer placed a new order for multiple products, so we want to insert it into our database making sure order ID, product ID, customer ID, product amount and order date are not null. All IDs must be unique.
Implementation of business rules: Use INSERT statement to add details of a new order.

2) **Imagined scenario:** The customer had a change of heart and no longer wanted the products, so they cancelled the order.
Implementation of business rules: Use DELETE statement with WHERE condition to delete the order using order ID and customer ID.

3) **Imagined scenario:** The customer claimed that they ordered a specific necklace and complained that they got the wrong one, so we wanted to search what they ordered so that we knew whether they were lying or telling the truth. In the event of them being right, we would need to apologize and either give them a refund or the product of their choice.
Implementation of business rules: Use SELECT statement with WHERE condition to search for the customer ID and order ID along with their order details.

Supplier Management

1) **Imagined scenario:** We made a new contract with a new supplier for a new product, so we need to store the new data in our database.
Implementation of business rules: Use INSERT statement to add new details of a supplier making sure supplier ID, supplier name, product ID and supplier contact number are not null. Supplier ID and product ID must be unique.

2) **Imagined scenario:** Our contract with a specific supplier expired and we no longer want to extend our contract, so we remove the supplier from our database.
Implementation of business rules: Use DELETE statement to remove the details of the supplier using supplier ID.

3) **Imagined scenario:** The contact number of one of our suppliers changed so we need to update it in our database.
Implementation of business rules: Use UPDATE statement to update the phone number of our

supplier making sure all other details are stored correctly and that none of them are null.

4) **Imagined scenario:** We want to contact one of our suppliers because their delivery of the product is past the due date, so we need to look up their contact number.
Implementation of business rules: Use SELECT statement with WHERE condition to select our target supplier with supplier ID.