Final Project CSC 4402

ECommerce Database

Christopher Duong

Patrick Nguyen

Reza Namakian

Josh Bankston

To-Do

Introduction

* Will contain the business description and sub-details (Project Objectives and Business Requirements)

Domain Application

* Where we will provide more in-depth details of our project including the different entities and relationships, what attributes they hold, and how we choose to implement everything
* List any assumptions or constraints on our entities
* ER diagrams, additional fields,

Database Design

* Show entity tables including relevant info such as primary key, field domains, etc for each
* Describe how we decide to make the database more efficient (ie. using normalization to reduce redundancies or ensuring data is being stored logically)
* BCNF or Boyce-Codd Normal Form is an end state of a database where if all data is in this form, then redundancies based on dependencies have been removed but others can still remain (See sample final report for examples of this)

Schema

* Diagrams depicting relations between tables (can show before and after redundancy/efficiency cleanup)
* Can describe how we collected data, but our case may just be manual input

Data Manipulation/Interesting Queries

* Where we display at LEAST 10 Select queries which demonstrate ample sql understanding
* Screenshots of queries and outputs

Conclusion

* Small blurb about what we have learned about databases/sql through the project

Table of Contents

Introduction

Databases are an integral part of any modern-day application or service and helps them to manage large amounts of information. Nowhere is this more prevalent than the world of eCommerce as can be seen with online retailers, ranging from huge corporations like Amazon to small-scale local stores. Inventory, sales, basic customer information, and order history are kept track of in order to provide valuable statistics for the company. Being able to monitor how specific items are performing in terms of sales and analyze the spending habits of customers to guide upper-level decisions are crucial to maximizing the growth of any online business. Our team aims to create a database that appropriately organizes relevant information such as item id’s, customer order history, and product info with their relations in a smart, efficient way.

Domain Application

The entities are:

* Products - (product\_id: int, name: varchar, description: text, category: int, store\_quantity: int)
* Price - (product\_id: int, original\_price: decimal, discount\_id: int, discounted\_price: decimal)
* Discount - (discount\_id: int, name: varchar, description: text, percent: decimal)
* Order\_items - (order\_id: int, product\_id: int, quantity: int)
* Orders - (order\_id: int, user\_id: int, total: decimal, created\_at: timestamp)
* Users - (user\_id: int, username: varchar, passwordHash: text, first\_name:

varchar, last\_name: varchar, created\_at: timestamp, country: varchar)

* Address\_book - (user\_id: int, address\_id: int)
* Address - (address\_id: int, address: varchar, city: varchar, country: varchar, phone: varchar)
* Payment\_book - (user\_id: int, payment\_id: int)
* Payment - (payment\_id: int, payment\_type: varchar, provider: varchar, account\_no: int, expire: date)

We created our database in SQLite and Txt format.

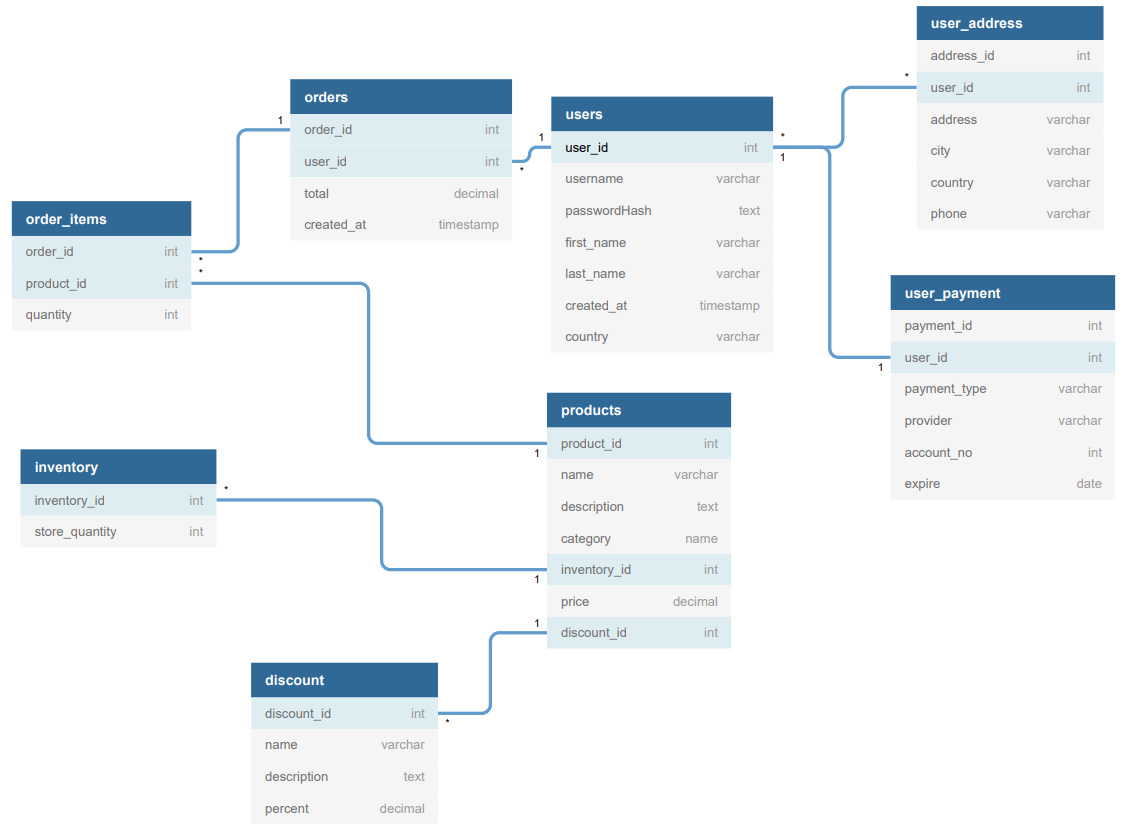
The assumptions we have:

* Each user has a unique user\_id
* There can be as many discounts applied to a product
* Only 1 payment type may be used to pay
* Each product has a unique product\_id

The constraints we have:

* 1 user needs to have 1 user payment
* There needs to at least have 1 quantity of a product to be placed in an order

ER Diagram:



Unique Entities:

* Users in user\_payment: each user payment has 1 unique user, user cannot have 2 payments and there can’t be 2 users for 1 payment.

Database Design

Tables of our entities, underlined attributes are primary keys:

|  |  |
| --- | --- |
| **Order\_items** | |
| order\_id | int |
| product\_id | int |
| quantity | int |

|  |  |
| --- | --- |
| **Orders** | |
| order\_id | int |
| user\_id | int |
| total | decimal |
| created\_at | timestamp |

|  |  |
| --- | --- |
| **Price** | |
| product\_id | int |
| original\_price | decimal |
| discount\_id | int |
| discounted\_price | decimal |

|  |  |
| --- | --- |
| **Products** | |
| product\_id | int |
| name | varchar |
| description | text |
| category | int |
| store\_quantity | int |

|  |  |
| --- | --- |
| **Discount** | |
| discount\_id | int |
| name | varchar |
| description | text |
| percent | decimal |

|  |  |
| --- | --- |
| **Users** | |
| user\_id | int |
| username | varchar |
| passwordHash | text |
| first\_name | varchar |
| last\_name | varchar |
| created\_at | timestamp |
| country | varchar |

|  |  |
| --- | --- |
| **Address\_book** | |
| user\_id | int |
| address\_id | int |

|  |  |
| --- | --- |
| **Address** | |
| address\_id | int |
| user\_id | int |
| address | varchar |
| city | varchar |
| country | varchar |
| phone | varchar |

|  |  |
| --- | --- |
| **Payment\_book** | |
| user\_id | int |
| payment\_id | int |

|  |  |
| --- | --- |
| **Payment** | |
| payment\_id | int |
| user\_id | int |
| payment\_type | varchar |
| provider | varchar |
| account\_no | int |
| expire | date |

Desired Normal Form

Desired normal form for each table

Schemas

Diagram of before and after decomposition/normalization methods

Select Queries

At least 10 select queries

* List all products with discounts
  + SELECT product\_id FROM price
  + WHERE percent > 0
* List all products with no discounts
  + SELECT product\_id FROM price
  + WHERE percent = 0
* List all products with category = electronics
  + SELECT product\_id FROM products
  + WHERE category = electronics
* List all users with at least 1 order
  + SELECT user\_id FROM users as u, orders as o
  + WHERE u.user\_id = o.user\_id AND
  + COUNT(o.order\_id) > 1
* List all users whose order total > 50
  + SELECT user\_id FROM orders
  + WHERE total > 50
* List all users whose address is outside of the United States
  + SELECT user\_id FROM address\_book inner join address
  + WHERE country <> “United States”
* List all orders which were made before 12-01-2012
  + SELECT order\_id FROM orders
  + WHERE created\_at < '12-01-2012 00:00:00'

Conclusion

Proper database design is vital to not only the longevity and consistency of the data being stored, but it is equally as important for scalability.

Decreasing redundancies helps not only to maintain consistency and accuracy of the data, but it also plays an important role in being able to edit data separately from other tables.

Scalability, longevity, sustainability, accuracy, security, and consistency