Introduction

Online ordering platforms have been a lifesaver for people in the United States where stores are extremely far apart and usually requiring a person to have a car to go to these stores since most of the time it is not practical to take public transport for carrying a lot of groceries. The pandemic also highlighted the crucial need for these platforms, as most people were unable to visit stores in person and had to rely solely on online platforms for their basic needs. Hence, for our project, we intend to build a database that tracks customer activities when using online shopping platforms. The differentiating factor between our database and that of any existing database of an online shopping platform is that we intend to track the activities of customers that are students to understand what kind of products they like to buy, the frequency of shopping, etc. It maintains records of the sales, customer details, information about the different stores that are a part of the platform, delivery, payment etc. This kind of database not only helps us know the target audience but also the current market trends based on a geographic location.

Logical Design

ERD Design

We went about designing the final version of our ERD keeping the following things in mind:

- A customer places an order which can consist of one or more products. Each of these products can come from any of the stores (or multiple stores). (Because this platform is designed for students, we took the concept of "Campus Pantry" into account. Hence, students just can see products on the online shelves but not the source of those products)
- Some products have a discount code associated with them whereas some do not. We also assumed that a particular discount can be applicable to multiple products (e.g., 20% off can be applicable to Apples and Bananas)
- We also assumed that a single order would have a single assigned delivery and a delivery agent can be assigned to multiple deliveries.

Tables and Cardinality:

1. **Customer**: This table has all the relevant information about a customer who is logging into the application to place an order. Since each customer can place multiple orders there is a one-to-many relationship between Customer and Order table. The relationship is non-identifying as the customerID in the Order table is a foreign key.
2. **Payment**: This table consists of the various payment methods a customer can use to make purchases and a unique identifier to identify them (PaymentID). Since a single payment method can be used for multiple orders a many to one relationship exists between Payment and Order table.The relationship is non-identifying as the PaymentID in the Order table is a foreign key.
3. **Order**: This order table has transactions made by customers for orders placed. Apart from the relationships between Payment and Customer tables, the Order table has a one-to-one relationship with the Delivery table. Since we have assumed that each order is linked to one delivery there is a one-to-one, non-identifying relationship using OrderID which is a foreign key in the Delivery table rather than being the primary key just like the Order table. An order can have multiple products and a product can be a part of multiple orders. Hence, there is a many-to-many relationship with the Product Table. It is created using a linking Order_has_Product table.
4. **Order_has_Product**: The Order_has_Product table acts as a linking table between Order and Product table and has both OrderID and ProductID as its primary keys along with quantity which is an attribute that depends both on the order and the number of products in that order. There is a one-to-many identifying relationship using Order ID which is a primary key in the Order table and the intermediate table. From Order_has_product an identifying relationship again exists with the Product table using ProductID which is a primary key in both the intermediate and Product table.
5. **Delivery**: The delivery table has information regarding delivery address, ETA, agent assigned, fee etc. and a primary key to identify all these entries (DeliveryID). Since a single agent can deliver multiple orders, this table has a many to one, non-identifying relationship with the Delivery_Agent table using Delivery_AgentID which is foreign key in the Delivery table but a primary key in the Delivery_agent table.

6. **Delivery_Agent**: This table has information regarding agents in the form of their rating, phone number and names.
7. **Product_Category**: This table identifies which category a product belongs to (for e.g produce, dairy etc) using a primary key called Product_categoryID. Since multiple products can belong to a single category, it has a one to many, non-identifying relationship with the products table using Product_categoryID which is a foreign key in the Product table.
8. **Discount**: The Discount table contains different types of discounts available over products in the Product table with their details and a primary key to identify them (DiscountID). Since we have assumed that a single discount code can be applied to multiple products (or not be applied), a non-identifying one-to-many relationship is created between discount and product tables using DiscountID which is a foreign key in the Product table. (Without a mandatory one)
9. **Product**: Product table has all the products and their details along with a primary key to identify them (ProductID). Apart from the mentioned relationships with Discount and Product_Category tables, there is a many-to-many relationship with the Store table. Since a single product can be in multiple stores and a single store sells multiple products, store and product tables are linked using a linking table Store_has_product.
10. **Store_has_product**: The Store_has_Product table acts as a linking table between Store and Product table and has both StoreID and ProductID as its primary keys alongwith stock details which depend on the store and the number of products from it. There is a one-to-many identifying relationship using ProductID which is a primary key in the Product table and the intermediate table. From Store_has_product an identifying relationship again exists with the Store table using StoreID which is a primary key in both the intermediate table and Store table.
11. **Store**: Store table consists of the stores from where the products are bought from along with their details like address and phone number.

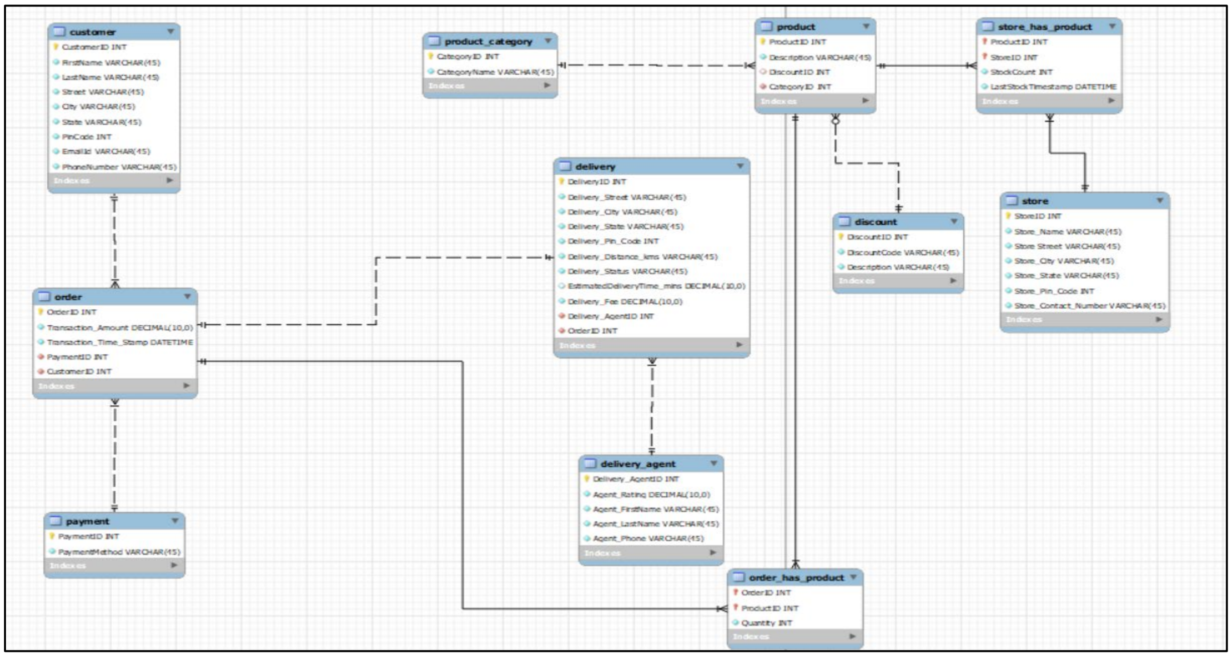

Fig 1. Final ERD

<u>Normalization</u>

1. **1NF:** We ensured that all the tables had a primary key that uniquely identifies all the entries in the table and also ensured that all the values are atomic in nature. This was done by splitting the Name fields into 'FirstName' and 'LastName' and the Address field into 'Street', 'City', 'State' and 'PinCode'.
2. **2NF:** Second Normal form states that all the non-key attributes must depend on the entire primary key. This was ensured in all the tables. For e.g., the 'StockCount' attribute in the 'store_has_product' table is calculated by counting the number of items of a particular product in each store. Therefore, this value depends on both the productID and storeID. Similarly, 'LastStockTimestamp' depends on the product and the store it belongs to. In this way all the attributes in the 'store_has_product' table depend on the entire composite primary key (productID and storeID) and the table is in 2NF.
3. **3NF:** Third normal form states that none of the attributes should have transitive dependency. All the non-key attributes in any table depend only on the primary key and nothing else. For e.g., we separated the attributes of delivery_agent from the delivery table and created a separate table since the agent attributes is independent of operational attributes like ETA, distance, etc.

Physical Database Design

1. After finalizing the ERD, we 'Forward Engineered' it to create a database with all the tables.
2. We did a quick review to check whether all the attributes and keys were in order and then started adding data to the tables.
3. The data for each of the tables was collected in Excel files and then imported directly into the tables. We had to ensure that the data in the excel files correctly matched that of the attributes defined in the tables.
4. We created a SQL script by exporting the database and added the CREATE DATABASE command. This script contained all the commands to create the tables and populate it with values. It was then run on different systems and tested.

Sample Data

- We took sample datasets from Kaggle that closely aligned with each of our tables such as Customer, Order, Payment etc.
- We modified the fields and records using Excel formulas and discussion amongst the team to suit the kind of customers, products and stores we had in mind to fulfill our objective.

Views and Queries

1. What are the least popular categories of products sold? - This query tries to identify our customer (students) behavior in terms of the products they prefer frequently and how this behavior is different from any other customer (e.g., a single/married person).
2. Which retail stores have the most sales? - This query tries to identify where students usually prefer to shop from. Since a student will also usually have monetary constraints, we can also find the cheapest stores in town.
3. Which payment ID is most and least preferred by students? - This query can shed light on why students might prefer certain payment methods. (e.g., many students do not own a credit card because there is no stable source of income yet)
4. Calculate the total number of discounted and undiscounted products purchased and identify which were purchased more? - Through this query we wanted to see if having monetary constraints gets in the way of a student's day-to-day purchases

5. What is the total number of delivery agents in each state? How do these values compare to the number of delivery agents having maximum rating in each of those states? - We wanted to see if the number of delivery agents is directly proportional to the size of state and if the rating is proportional to the number of delivery agents in the respective state.
6. How many deliveries were 'COMPLETED' where the estimated delivery time (EDT) is greater than the average EDT of all the completed deliveries? - We wanted to observe the delivery statistics for our online ordering platform based on our student customers and their location of delivery. Students usually stay in areas where housing is affordable, but this can be far from the busier areas with stores, offices etc.
7. Which state has the most delayed deliveries (wherein the EDT is greater than the average EDT)? - We wanted to see among all the deliveries how many were delivered late and wanted to observe how effective the delivery system is. This also helped with understanding if the deliveries were late because of agents being busy or delivery location etc.

Views/Queries	Req. A	Req. B	Req. C	Req. D	Req. E
Q1) Least_Popular_Categories	X		X	X	
Q2) Top_Sales_Stores	X		X	X	
Q3) Payment_Methods	X				
Q4) Product_Quantities	X		X		X
Q5) Delivery_Agents_States	X	X	X		X
Q6) Delayed_Deliveries		X	X		X
Q7) States_Delayed_Deliveries		X	X		X

Table 1. Requirement satisfaction for queries

Changes from Original Design

1. Our initial ERD had 9 databases. In the final one we had 10 tables to include information about stores, delivery agents, their linking tables etc.
2. Removed certain tables with one-to-one relationships and merged their attributes to one table.
3. Normalized the tables by splitting multi-valued attributes and adding the right attributes to the right tables. For e.g., ‘Quantity’ attribute was removed from ‘Order’ table and added into ‘Order_has_product’ table since it depends on both the order and the count of product.
4. Changed data types of fields into the appropriate types. For e.g., ‘EstimatedDeliveryTime’, ‘LastStockTimestamp’ was changed to datetime from decimal.

Issues and Solutions

- Vast scope of the Project: Deciding the scope of the project was an issue since we aimed to develop a good project without making it too complicated for everyone to understand. For e.g., our initial model had a separate member field which was supposed to indicate repeat customers of the app who had created an account and only they would have their address, email id and other details saved. The other one-time users would not have to enter these personal details. This idea was later discarded as it would lead to a lot of values with null entries in the ‘Customer’ table.
- Incorrect normalization: Creating the ERD was challenging as we encountered problems that included cardinality issues, incorrect normalization, and other inconsistencies. For e.g initially, the ‘Quantity’ field was just in the ‘Order’ table. This was incorrect since it was not obeying the third normal form as ‘Quantity’ depends not just on the order but also the number of products ordered. The issue was fixed by moving it into the ‘order_has_product’ linking table.
- Several one-to-one relationships: Our initial ERD had multiple tables with one-to-one relationships. We restructured these tables to include all the related attributes into a single table. For e.g., the ‘Delivery’ and ‘Delivery_Details’ table had fields that did not require separate tables.
- Match data from excel files with data type in Database: Some fields such as the ones containing timestamps did not import at the first attempt. We realized that the issue was with the datetime format. In MySQL it is ‘YYYY-MM-DD hh:mm:ss’ whereas Excel, by default, stores it as ‘MM/DD/YYYY hh:mm’. We had to reformat the dates and then import it into the

database. A blank cell in Excel by default does not get imported as a NULL value in MySQL. We had to type NULL in all the blank cells of our excel data for it to get imported correctly.

- Methods of collecting data: We initially considered doing a survey to get some representative information about students' online ordering patterns. However, we later realized that most students might not be willing to give personal information such as the products they ordered, their address etc. Also, most of them would probably not have the time to fill out a survey with all these details. Therefore, we decided to take the sample data from relevant Kaggle datasets and modify it appropriately to fit in our tables.

Lessons Learnt

- Importance of proper Normalization: We learnt how to properly analyze which attributes depend on each other and normalize the tables in a database. Normalization is crucial to ensure data integrity by enforcing rules that prevent invalid data from being entered into the database.
- Proper planning and organization are crucial: We realized the importance of defining the scope and goals of the project right at the beginning and following it. Sometimes we had to go back and forth between the ERD and the database creation since we would catch some errors or change the logical design, this could have been avoided with better planning.
- Data Integrity: It is important to maintain data integrity and make sure that all the entries match the data types of the relevant fields.
- Communication is key: Collaborating with team members and communicating effectively is essential for completing any project successfully. With respect to this project, since there were different parts that had to come together in the end (finalizing an ERD, sample data creation, running queries, testing etc.) it was important that all of us were on the same page and working together.

Strengths of Design

- We tried to make our queries in a way that gives insights that can be useful for an organization. For e.g.one of our queries tries to co-relate the total number of delivery agents in a state to the number of delivery agents that are highly rated.
- We tried to populate all our tables with a large number of diverse entries to get a representative result.

- Our final ERD design tries to incorporate all the fundamentals of an online delivery system while still being easy-to-understand and not too convoluted. We verified the ERD with the TA multiple times to match all the requirements and expectations set for the project.

Limitations of Design

- Currently, we don't have a relationship between Order and Store Tables signifying the fact that customers do not have an option to choose a store to buy products from. The store Table is connected to the Product table only.
- We didn't show any records of a failed delivery in the Delivery table and a consequent successful re-attempt for that delivery. This would have increased the design complexity.

Future Extensions

- App Development: We can create a delivery app by creating a front-end application and integrating it with our back-end database.
- Including Store option: For the project we tried to resonate UMD's pantry and Instacart. Customers currently just order the products without knowing which store they are getting it from. But in the future, with a bigger customer base we can provide the customer with the option to choose from which store they want to order their products.
- Increasing Data collection: For the scope of the project, we limited the data collection to three states: CO, CA, MD. In the future we can increase the data, collecting the information of customers in different states.

Alternate implementation

- Most popular online grocery delivery platforms use PostgreSQL as their primary database management system. These platforms require large volumes of data, need to support complex queries, and provide high availability.
- PostgreSQL is designed to be scalable and can run multiple queries concurrently making it better suited for large scale applications than something like MySQL. Another benefit over MySQL is that it uses object-oriented programming, which allows for user-defined types, inheritance, and complex data structures.