Scenario #7: The Smelly Mask Company

Team Nevada:
Andres Toro, Derek Stayoch, Dan Vo, Favour Salako, Diego Soto, Johnathan Tait, Felipe Silva

Background

- The president of The Smelly Mask Company, had a realization about the problem of people forgetting to wear face masks during the Covid-19 pandemic. This inspired him to start a company that initially offered scented masks under the name "Floral Gardens Mask Company." However, they found that even pleasant-smelling masks didn't solve the issue. Chris discovered that using a powerful negative motivation tool, making the masks smell bad initially, prompted customers to remember to wear them!
- The Smelly Mask Company started experiencing high demand for their products. They required a database to manage customer information and orders. Therefore, the program needs to collect customer data, offer discounts to repeat customers, track odorous orders with specific smells and colors, handle returns with unique codes and record-keeping, and send survey forms to understand product returns.

Team Members

1. The Database Administrator (Dan Vo) - be responsible for describing the database that the team decided to create.
2. The Database Developer (Dan Vo) - determine what types of processing will be needed by the customer that the DBMS.
3. The Database Tester (Felipe Silva) - describe their plans for maintaining database integrity.
4. Database Analyst (Favour Salako) - determine what scheduled procedures, stored functions, and triggers will be required.
5. Database Engineer (Diego Soto) - explain how the team's database was normalized.
6. Database Architect (Andres Toro) - work with the database modeler to create a design via the ER model.
7. Database Modeler (Johnathan Tait) - responsible for creating an ER diagram of the database that the team decides to build.

Database Architect

We use 8 steps to define what our goals are for the Database system

1. Identify entities: Customers, Orders, Inventory, Returns
2. Define attributes: Our attributes will be necessary information for each entity such as customer information, descriptions of products and how much is available. As well as dates and booleans involving returns.
3. Establish Relationships: Customers table relate with Orders table based off customer ID. and the Orders table uses Order Id for Returns table and quantity for inventory table.
4. Assign Cardinality: One customer can have many, orders but each order can only have one return associated with it. Also, inventory can be related to more than one order.

Database Architect

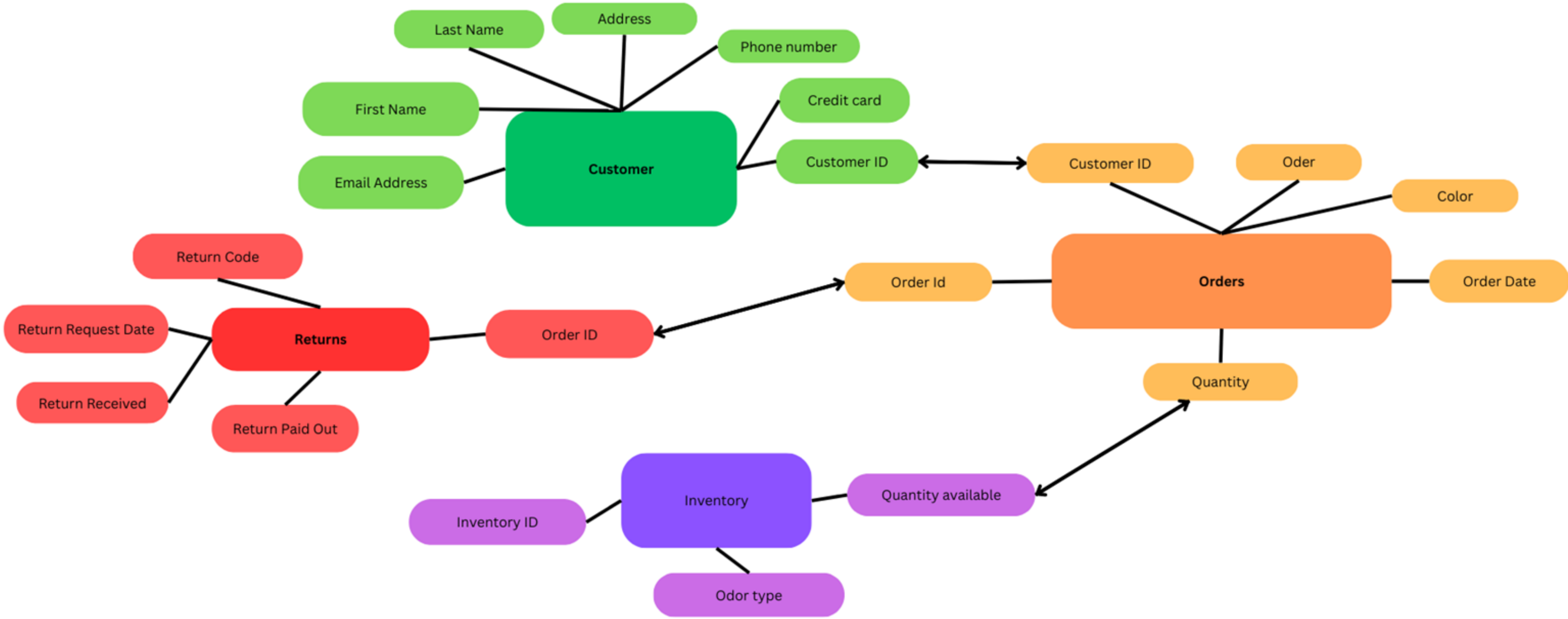
5. Normalize the Model: We can make our tables into 1st Normal Form by requiring each table to have at least one Primary key.

6. Refine Attribute Definitions: We make sure that each data type is appropriate for each row and we are not taking up memory unnecessarily.

7. Validate and Iterate: We will make sure that we make PK set to not null, that way we can maintain data integrity and prevent duplicates in our Database.

8. Document and Model: Using a ER diagram we can better illustrate how the tables in our database all work together to provide a better ordering and inventory system.

Database Modeler



Database Engineer

- The database needed to be normalized, and it was done in various ways.
- This was done in 1NF (First Normal Form), since the rows and columns used in all tables contain no duplicate values.
- Each data is unique from one another, and therefore redundancy is minimized each table contains data that isn’t so repetitive.
- Redundancy is maintained since there are three primary keys throughout this database, which are the following: customerID, inventoryID, and orderID.

Database Engineer

- Joins be used to extract important information from the database:
    - SELECT firstName, lastName, odor, quantity FROM customers c JOIN orders o ON c.customerID = o.customerID. Retrieves customer information based on orders.
    - SELECT * FROM returns r JOIN orders o ON r.orderID = o.orderID. Shows information about returns on given orders.

Database Analyst

Functions:

- **random _Return_no(id)** -> This returns a random integer return_number once ran, and inserts it in the returns table where the id =@id(parameter)
- **Email_sender(email_address)** -> Taking the customer’s email as a parameter, this function sends an email to said customer

Triggers:

- **Initiate_return** -> In this trigger, when a return is initiated (A new returnID is inserted), the random_Return_no() function runs
- **Complete return** -> in this trigger, when a return has been received and refund issued(returnReceived & refundPaidOut columns in the returns table becomes ‘T’), the Email_sender() function runs

Database Analyst

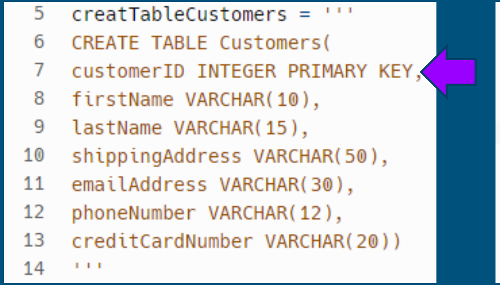
The database will need some triggers, stored functions and sometimes stored procedures to function effectively.

Procedures:

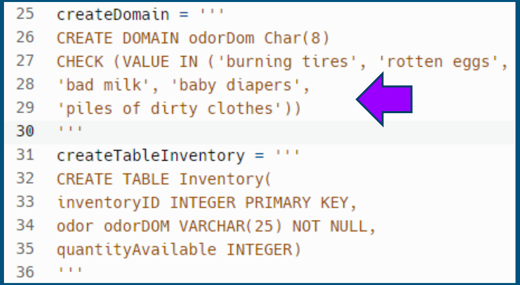
- **Cust_order** -> Select odor, quality, color from inventory where order = @order and color = @color. We use this to allow customers order masks based on odor, quality & color .
- **Cust_check** -> Here we check if the customer exists in the Customer Table and offer a 10% discount on their order total, else Insert the customers Information into the table

Database Tester

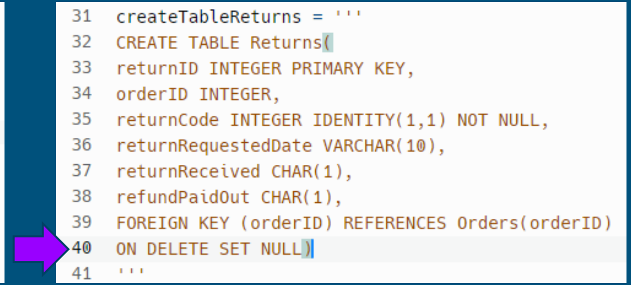
- Ensure the database maintains its data integrity.
  - Consistency checking using keys and length limits.
  - Entity integrity.
  - Domain integrity.
  - Referential integrity.
  - Temporal data.



```
creatTableCustomers = '''
CREATE TABLE Customers(
customerID INTEGER PRIMARY KEY,
firstName VARCHAR(10),
lastName VARCHAR(15),
shippingAddress VARCHAR(50),
emailAddress VARCHAR(30),
phoneNumber VARCHAR(12),
creditCardNumber VARCHAR(20))
'''
```



```
createDomain = '''
CREATE DOMAIN odorDom Char(8)
CHECK (VALUE IN ('burning tires', 'rotten eggs',
'bad milk', 'baby diapers',
'piles of dirty clothes'))
'''
createTableInventory = '''
CREATE TABLE Inventory(
inventoryID INTEGER PRIMARY KEY,
odor odorDOM VARCHAR(25) NOT NULL,
quantityAvailable INTEGER)
'''
```



```
createTableReturns = '''
CREATE TABLE Returns(
returnID INTEGER PRIMARY KEY,
orderID INTEGER,
returnCode INTEGER IDENTITY(1,1) NOT NULL,
returnRequestedDate VARCHAR(10),
returnReceived CHAR(1),
refundPaidOut CHAR(1),
FOREIGN KEY (orderID) REFERENCES Orders(orderID)
ON DELETE SET NULL)
'''
```

Database Developer

Data Processing:

- Process Customer Sentiment
- Personalized Recommendations

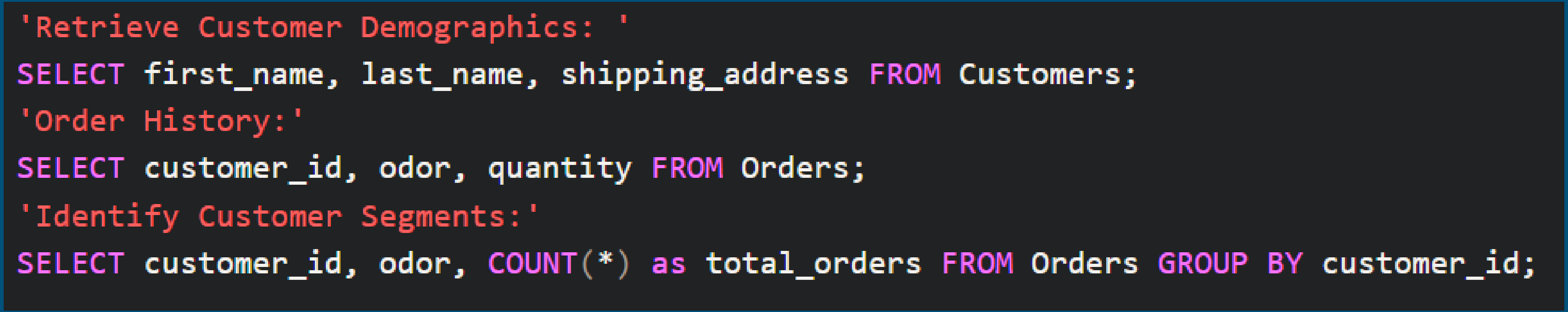
Data Addition & Modification:

- DBMS inserts data into each table (Customer, Orders, Inventory, Returns) and their respective columns
- Modifying Inventory Quantity and/or Returns and Refunds

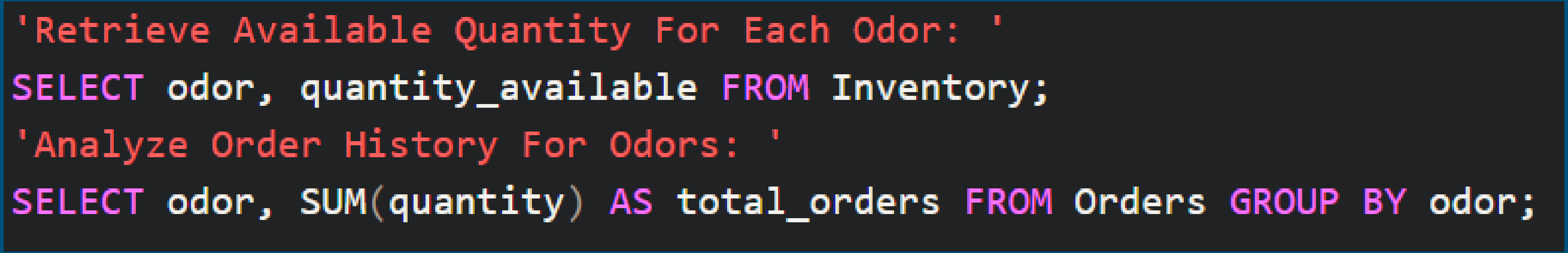
Data Extraction & Use:

- Inventory management: Extract data from Inventory table to monitor available quantity of each color, track inventory levels, and generate restocking alerts (it could be used in junction with triggers or thresholds)
- Customer Sentiment: Extracting data such as order history, return requests, can aid with processing into customer preferences and satisfaction.

Database Developer



```sql
'Retrieve Customer Demographics: '
SELECT first_name, last_name, shipping_address FROM Customers;
'Order History:'
SELECT customer_id, odor, quantity FROM Orders;
'Identify Customer Segments:'
SELECT customer_id, odor, COUNT(*) as total_orders FROM Orders GROUP BY customer_id;
```



```sql
'Retrieve Available Quantity For Each Odor: '
SELECT odor, quantity_available FROM Inventory;
'Analyze Order History For Odors: '
SELECT odor, SUM(quantity) AS total_orders FROM Orders GROUP BY odor;
```

Database Administrator

We will need to create a database that consists of at least 4 tables that include

- Customer Information
- Order Information
- Inventory Information
- Return Information

Each table should have a purpose in our database and should all have at least one or more relations to each other.

Database Administrator

- Customer Information should contain a field that relates to order information, not only for inventory information, but to see if customers qualify for returning customer discounts so we can get repeat buyers.
- Return and Order information tables should have a strong relation because we can not have a return without an order to begin with.