BorderGuru – Full Stack Developer

Architectural Design:

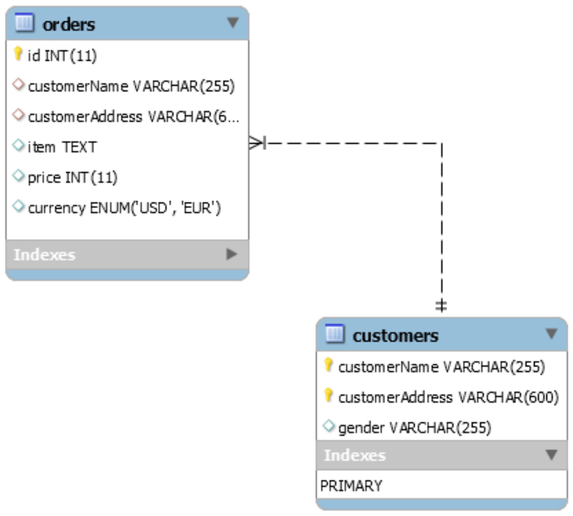
There are two tables in the database

1. Orders
2. Customers

Customer table contains all the customer attributes. Customer table comes up with system defined customerName and customerAddress attributes. And APIs've been provided to add as much attributes as you want for the customer schema. customerName and customerAddress both are being used as unique composite key for the customer table to uniquely identify a customer. I assume that customerName and customerAddress is unique for each customer.

Orders table contains all the order attributes along with customerName and customerAddress. To support backward compatibility, database is designed in a way to fulfill all the needs that stakeholders can demand in the future. customerName and customerAddress both make up a foreign key in this table and have **ON UPDATE CASCADE** and **ON DELETE SET NULL** as foreign key constraints. This is to make sure that order table structure is not modified and is backward compatible also. So, now when in future, we have an order with some additional information regarding customer, we can store than in customer table and link to order table with customerName and customerAddress attributes. This does not modify the order table structure and adds the freedom to store new customer attributes.

If I had the option to modify the schema I'd definitely have used customer id in orders table but due to that restriction this was the best design I would come up with to work with previous design and support backward compatibility when new information related to customer is supposed to be added in the system.



orders
id INT(11)
customerName VARCHAR(255)
customerAddress VARCHAR(6...
item TEXT
price INT(11)
currency ENUM('USD', 'EUR')
Indexes
customers
customerName VARCHAR(255)
customerAddress VARCHAR(600)
gender VARCHAR(255)
Indexes
PRIMARY

Question: Why did you pick your first particular your design? What assumptions did you make, and what tradeoffs did you consider?

Description. In the first part I assume that the customer for which the order is being placed already exists in the system. If you see the UI part, I've designed frontend in a way where you select a customer from dropdown. And dropdown has a list of customer defined/saved in the system. If there is a need to place an order for a customer who is not in our system or a new customer for example, first, you'll have to add that customer in the system and then select that customer from dropdown, fill the other order attributes and save the order. And I assume customerName and customerAddress are unique for each customer. I'd have stored customer ID in this table rather than customer name and address but due to restriction of not modifying the order table structure and backward compatibility, this was not a good idea.

Question: What did you consider when you had to design the second solution? And which assumptions and tradeoffs you made?

Description. First let me say that this was the tricky part of the assignment. For this, I had to design the db in a way that my db queries don't have to change and would work even on adding more information related to customers without modifying order table structure.

First I thought of using NoSql database like MongoDb for this. But that would modify order table and I'd have to do a lot to deal with managing new customer information like, how to get to know what attribute belongs to customer only. I could not store it as customer object because already stored information was not in that format. And If I store this information in a separate collection that won't be a good idea because mongodDb fits best where there is raw data and has minimum relations between. That left me with an option to use some structural database and finally I decided to use **mysql**.

So, I used the previous order structure in way that helped me out to support this customization. So, I used customer attributes of order table as link between the new table for new customer attributes (customer table). For this I assume that the customer for which the order is being placed already exists in the database. In other words, first you'll save customer in the database and then place his order putting customerName and customerAddress along with other order attributes. Again, I assume that customerName and customerAddress is unique for each customer.

Question: What do you like (and dislike) about Node/Javascript compared to other programming languages?

Description. Well, to be honest, I love Javascript. And NodeJs simply gives support to this interest on backend. So, I really enjoy working with NodeJs. I also enjoy working with Java and other Java frameworks like Grails but yeah I enjoy working with NodeJs as it is the best solution when you have network intensive application, it uses Javascript and has a huge ecosystem of open source libraries.

Note:

- For the time being, only two type of currencies are supported, EUR and USD. But can easily be customized to add the support for new currencies.
- Default port for application is 3000. Make sure that port is available if you don’t have an environment variable **“PORT”**.
- To know how to run and what’s in the application, please go through the ReadMe file added in the application folder.