

NeoSoft Technology

Database Design

September 09, 2017



NeoStore

Database Design for NeoStore Project

In this doc, we are going to just discuss the overview and design of the database which is not specific to any database.using this design you can work with any database and create the backend.

In the NeoStore project, we have used MongoDB for database and loopback framework for interacting with the database and on the frontend for admin panel and users websites we have used angular2.

Name Of Tables:-

1.user_account.

2.category.

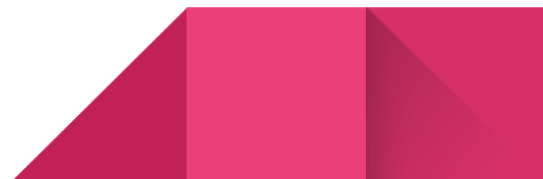
3.product.

4.shoppingcart.

5.image.

6.order.

7.orderitems.




8.colors.

9.address.

10.rating.

As you can see above are the table used in the project relationship between table is made to manage the data in the table and segregate the data as much as possible to avoid data redundancy.Below figure shows you schema of the entire database.

Also, keep in mind if you are using any database with the framework then you can have two types of the query in your application in which one query will be database specific while another is framework specific, Both queries does same work and gives same response just the syntax of writing both are different.The advantage of a framework-specific query is that if in the case in future your database might get change then too you do not need to change the query in the entire application.



DataBase Schema:-

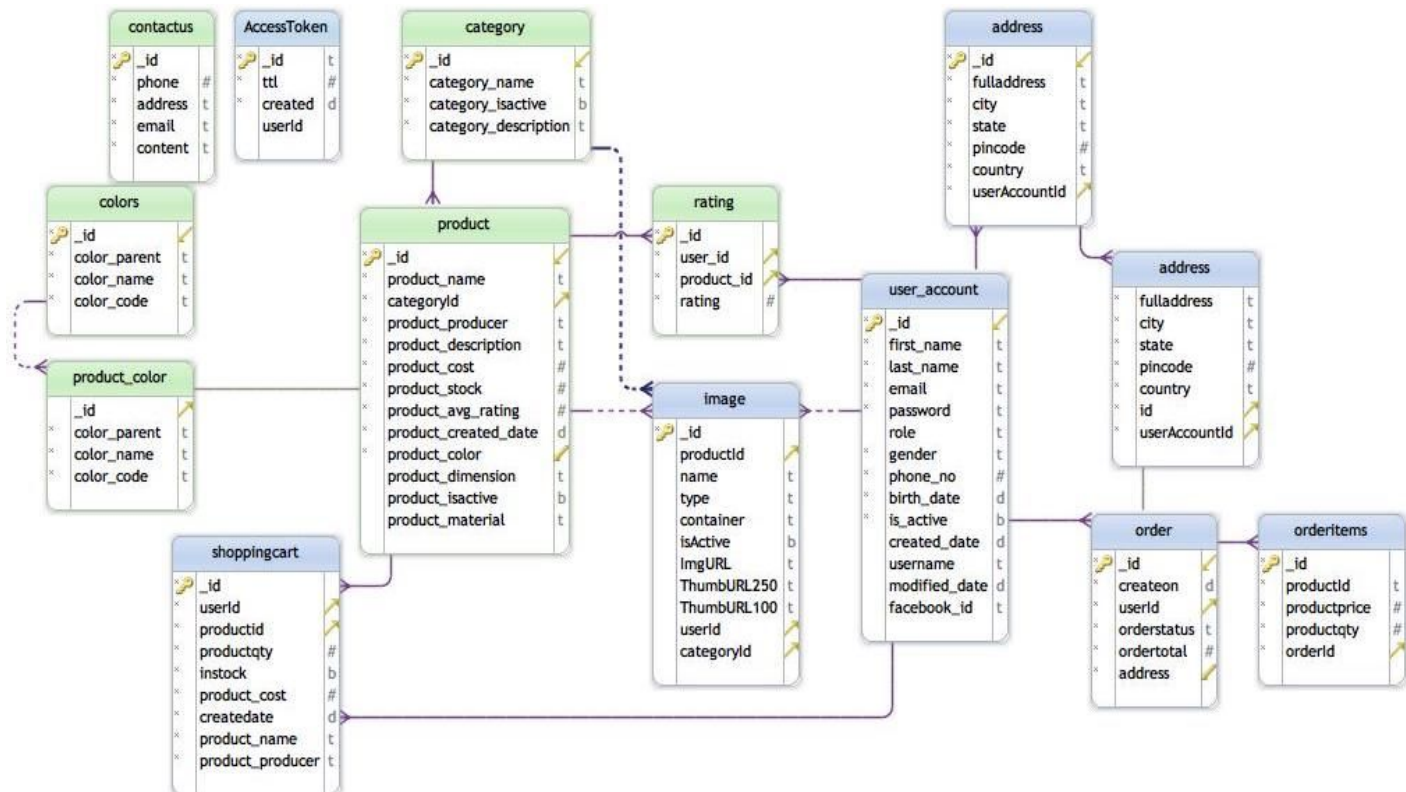


Fig: DataBase Schema

The above figure shows the database schema used in neostore. In this, we have used the table as indicated in the table list. Here we have segregated the data as much as possible to avoid data redundancy.

Starting with user_account table, it has a one-to-one relationship with image table so that even multiple images are uploaded for a user, the image that is uploaded latest is taken. Also, we have made a custom API for images upload of product, category, and user profile pic. user_account has one-to-many relationships with address table as a user can have multiple addresses and user_account has one-to-many relationships with order table as a user can have many orders. user_account has one-to-many relationships with shoppingcart table as a user can have many items in cart. user_account has one-to-many relationships with rating table as a user can give many ratings to products, but we have made a custom API if the same user rates the same product, then his rating gets updated. We have also maintained the concept of access token in our app; you may not be able to see its table in our app, but as we are using loopback, it itself manages all security regarding access token. If you are working with express or any other framework, you might take help of node packages "JWT" which might help you in maintaining access token concept. Also, for social login, we have implemented a custom API using which we can have login of a user using social credentials. For this, you can use "Passport" using which you can have social login into the app.

The category has one-to-many relationships with product table as a category can have multiple products and category has one-to-one relationship with image table as we have stored one image for each product.

Product table has one-to-many relationships with image table as we have stored multiple images for each product. Product table has one-to-many relationships with shoppingcart table as we have stored multiple products for each user in shoppingcart.



table. In product table, we have also entered color which comes from a color table we have done so to have filter mechanism for this in color table we have set of color from that set of color the only color is assigned to each product. a product has one to many relationships with rating table as a user can have give many rating to products.

Order table has one to many relationship with orderitem as order has many products so these product details are stored in orderitem table we have maintained separate table rather than array for segregation of data as much as possible it is better to avoid array because if you want to update any information in an array it becomes difficult to manipulate array so try to avoid array for eg: A user has many addresses so if you try to store address of user in an array in user_account table so if you want to edit an address in between an array it becomes difficult to manipulate the array so array in table must be avoided as much as possible. There we have the separate table for address and images. also it is necessary that quantity of product, cost of product and address selected by the user much not changes so we have orderitem table to store all this as the order is placed successfully. for order we have created a custom API in which we first check the card details enter by user using "STRIP" after successfully verifying their user's card details we generate his order in order table with his address and total cost after that we have product entry into order item table to have quantity and cost of product with product id. now user order id or userId you can obtain order details or product details for that order. We have one to many relationship between order as we store an entire address of user while placing the order.

We have the separate table for image for avoiding array in the table and also for images you need to have the thumbnail so to manage thumbnail with images we have the separate table for image.



