Local Line Technical Test

Non Coding Questions

Visibility Problem

In a system like Local Line where buyers of a specific region should only see the products of the suppliers in the same cities where the suppliers deliver, a search functionality is needed to show the correct products in a fast, reliable, accurate, and scalable way.

- Would you use a relational or a non-relational database architecture? Why?

- Explain what technologies would be needed to implement search functionality

Answer:

If we are to query data based on multiple filters or complex filters, then using a SQL database would be the ideal choice and we can use indexing for faster search results.

Here in our case, we would like to get products from suppliers based on specific cities/region. So, using a NoSQL database would be a better option as it offers very high performance and supports high scalability.

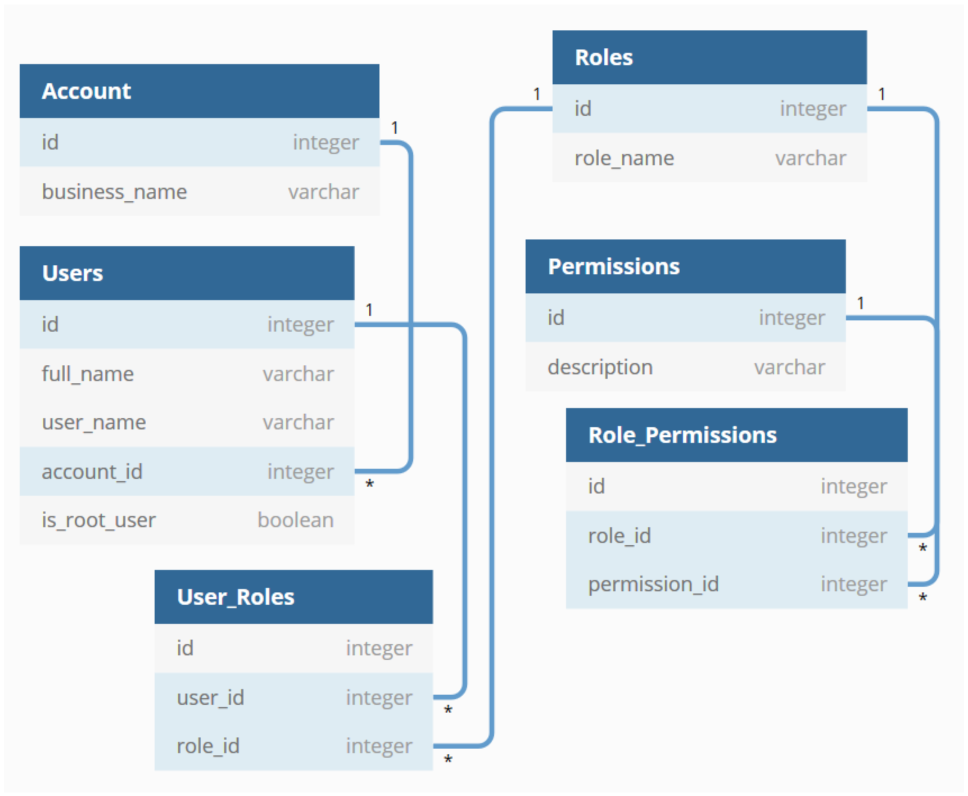
And if we do need to maintain structured data, then we can use ElasticSearch exclusively for searching. For example, when we insert data/transaction to the SQL database, the Backend (e.g. API) can save parts of the data which will be used for searching to ElasticSearch as well.

Multiple Logins per Business

We're planning to make a feature which allows each account to create multiple logins. Each account has a business name property. Each login will be associated with a different role and each role will have different permissions. The roles are flexible and can contain any set of permissions and should be managed only by the root user(s) in a business. Each account has at least one root user.

Provide a diagram outlining the best DB structure to store the hierarchy of the accounts in a RDBMS.

Solution:



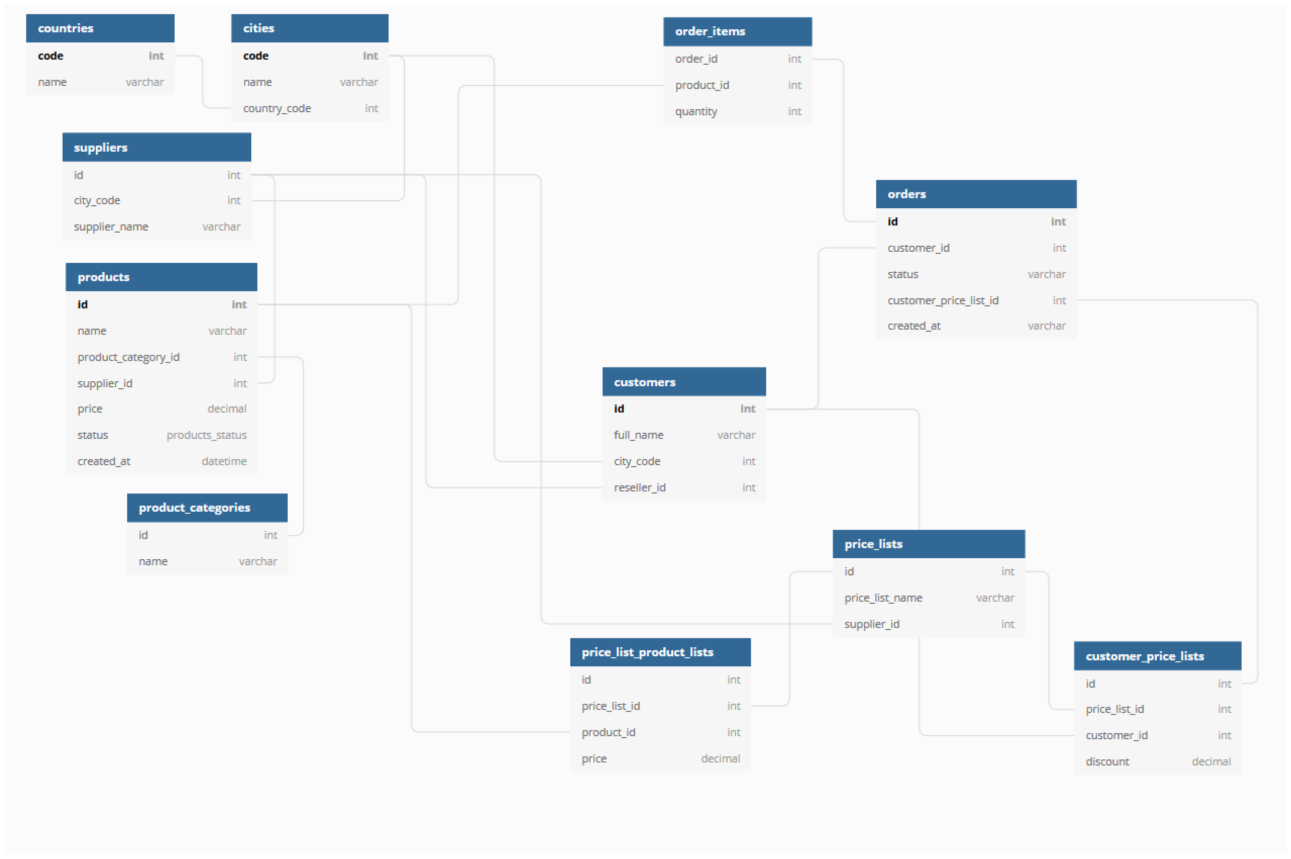
Product pricing, Categories and Discounts/Markups

On the Local Line platform, every supplier should be able to show a different price list for every customer (or group of customers) they may have. We expect the platform to serve tens of thousands of products per day.

Provide a diagram outlining the best DB structure for this in a RDBMS.

Bonus: Some suppliers on Local Line resell and/or repackage other suppliers' items. How would you fit this into your model?

Solution:



MVC Framework

You have a database with the following models:

`class Account():`
 	
`has_many Order`
	`class Order():`
	
	`has_many Item`
		`class Item():`
		
		`has_one Product`
		
		`int_quantity`
			`class Product():`
			
			`double_price`

You have a route that needs to do the following (given a user_id, buyer_id, seller_id, and order_id):

1. Check for user authentication check_auth(user_id)

2. Query the database for the required order

3. Calculate the grand total for that order calculate_grandtotal(...)

4. Return the result to the user

For each process, would you put it in a model or a controller? Why?

Answer:

1) It's a general and recommended practice to declare an "Authorize" attribute or create a Custom Authenticate filter and use it before the Controller definition, which means any request to the action methods in that Controller must have an Authentication Token. So, the authentication check will be performed for every single request using the default method in case of using a framework such as "oAuth" or the Custom filter in case of custom implementation.

2) and 3) It's recommended to have thin Controllers and it's a common practice to have a Service class defined where all Business logic could be performed, and a Data Access Layer Class which performs CRUD operations on the Database.

So, in this case, we can have a database view created with the following structure

Order_Id	Item_id	Product_Id	Quantity	Price

The Data Access Layer will have a function to fetch results (IEnumerable<CustomModel>) by passing the view name and the parameters (buyer_id, seller_id, and order_id). The CustomModel will match the same structure of the view.

The Service Layer will have a function similar to below,

```
public double calculate_grandtotal(buyer_id, seller_id, and order_id)

{

//Pass the parameters to the function in Data Access Layer

//Calculate the grand total from the resultset

//Return grand total to Controller

}
```

And the controller returns the grand total as a JSON object.