ALUCA

Sistema de Gerenciamento para Redes de Supermercados

Alunos: Caio Balbi, Luiz Alexandre, Maria Luísa

Descrição do problema

O cliente é uma rede de supermercados que precisa de um sistema para gerenciar o estoque e venda de seus produtos em cada uma de suas lojas, de forma que ele possa fazer um gerenciamento melhor dos produtos que ele compra para vender. O cliente deseja também, ter visibilidade dos seus funcionários, de forma que ele seja capaz de acompanhar o cargo, o salário e as horas trabalhadas de cada funcionário.

Para o cliente, é importante que ele possa visualizar os produtos que estão ficando sem estoque e com isso possa realizar o pedido de compra para um fornecedor, também pelo sistema.

O cliente deseja ter informações de horas trabalhadas de cada funcionário.

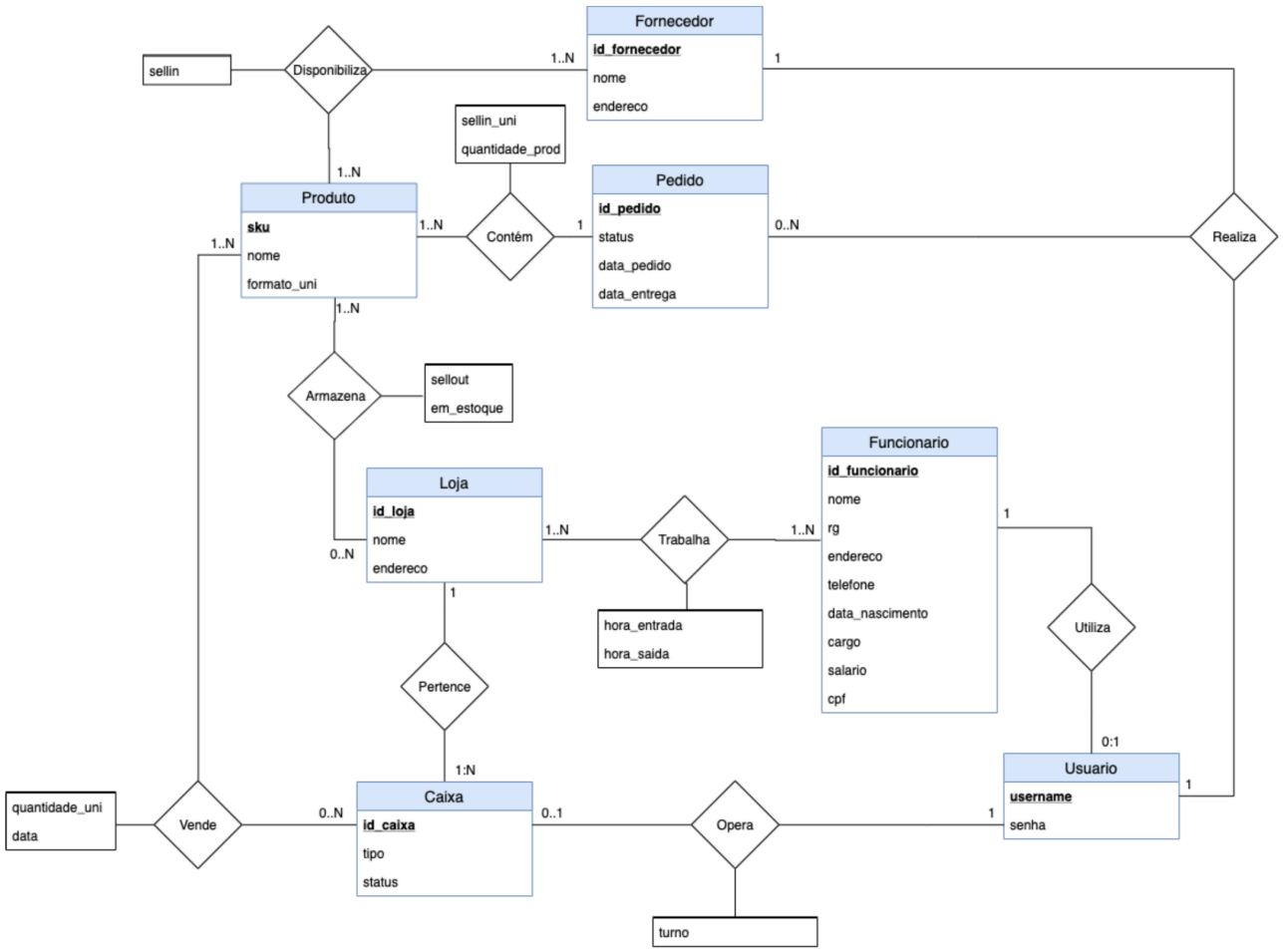
Informações do consumidor final dos produtos vendidos pelas lojas do supermercado **não** são relevantes neste sistema, uma vez que o cliente deseja apenas saber o volume de vendas de cada produto para obter informações de estoque.

Regras de Negócio

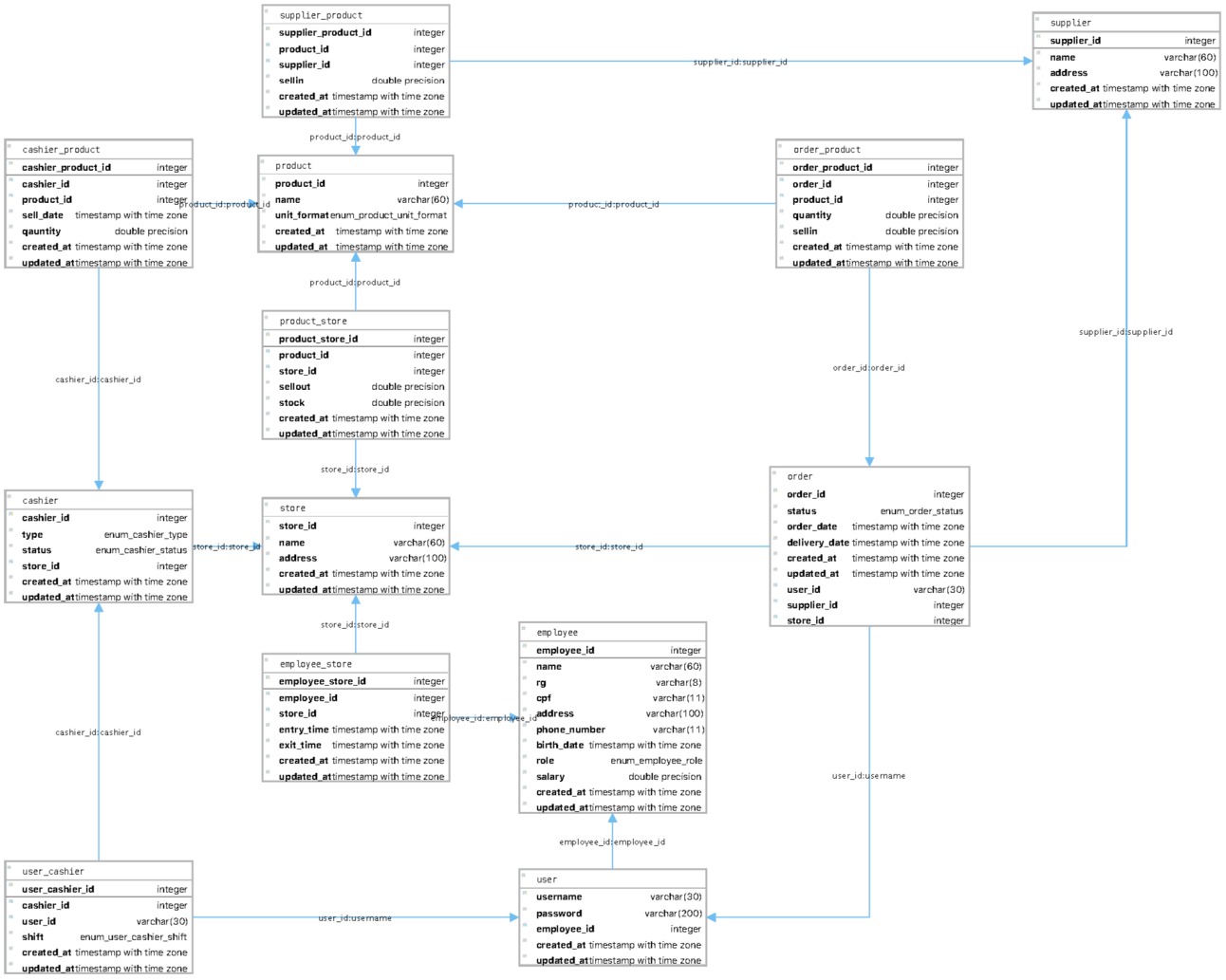
1. Os produtos serão identificados por um valor de *Stock Keeping Unit* (**SKU**), que é um valor interno gerado pelo sistema para diferenciar cada produto. Se temos um produto como *BANANA*, podemos ter diferentes SKUs como BANANA PRATA, BANANA NANICA e BANANA MAÇÃ, e cada SKU pode ser vendido a um preço diferente e teremos estoques diferentes para cada um deles
2. O sistema deve incluir registros de funcionários da rede de supermercado para que possamos ter registro de quantos funcionários existem, quais cargos esses funcionários ocupam, qual o salário dos funcionários e em quais lojas eles trabalham
3. Os funcionários podem ocupar os cargos: **embalador**, **operador de caixa**, **repositor**, **balconista**, **auxiliar**, **subgerente** e **gerente**
4. Apenas os funcionários que ocupam cargos de **subgerente**, **gerente**, **repositor** e **operador de caixa** podem ter usuários cadastrados no sistema
5. Os funcionários que ocupam cargos de **repositor** podem ou não ter usuários no sistema
6. Um **operador de caixa** pode operar caixas diferentes em turnos diferentes, o sistema deve manter sempre os turnos e timestamps de entrada e saída em que o funcionário trablhou na operação do caixa
7. Cada loja do supermercado precisa ter pelo menos um caixa

8. Funcionários que ocupam cargos de **gerente**, **subgerente** ou **repositor** podem realizar pedidos de compra no sistema para repor os produtos do estoque do supermercado
9. Consideramos como fornecedores, todas as entidades das quais o supermercado pode comprar produtos para repor o seu estoque. Por exemplo: centros de distribuição ou fábricas que realizam venda direta
10. No sistema, um mesmo produto pode ser fornecido por mais de um fornecedor diferente e cabe ao usuário que está realizando o pedido de compra escolher de qual fornecedor ele prefere comprar e qual a quantidade de produtos que será comprada
11. Cada loja armazena quantidades diferentes de cada produto
12. As lojas do supermercado podem vender um mesmo produto a preços diferentes
13. Um pedido de compra pode possuir os status **em espera**, **a caminho** ou **entregue**
14. O preço que o supermercado compra os produtos do fornecedor (sellin) pode ser diferente do preço pelo qual o supermercado vende o produto para o cliente final (sellout)
15. Não precisamos manter dados do cliente final no nosso banco, mas precisamos sabes as informações qual produto passou no scanner de cada caixa do supermercado

Modelo ER



Tabelas do Banco



Disponíveis no anexo `diagrama do banco.pdf`

Como rodar a aplicação

Pré-requisitos

- Docker version v18.09.1
- docker-compose version v1.23.2
- NodeJs v12.3.1
- Yarn v1.16.0

API+Banco - diretório bd-backend

Colocamos toda a parte do backend em containers que não fosse necessário instalar e configurar o banco na máquina. Aqui temos um problema com o `database.json` que não tivemos tempo de descobrir o que era, mas para realizar os comandos `make migrate-tables` e `make populate-tables` o `database.json` precisa ser

```
{
  "development": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "localhost",
    "dialect": "postgres",
    "define": {
      "timestamps": true
    }
  },
  "test": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "postgres",
    "dialect": "postgres"
  },
  "production": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "postgres",
    "dialect": "postgres"
  }
}
```

para rodar os demais comandos esse arquivo precisa ter o conteúdo:

```
{
  "development": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "postgres",
    "dialect": "postgres",
    "define": {
      "timestamps": true
    }
  },
  "test": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "postgres",
```

```
    "dialect": "postgres"
  },
  "production": {
    "username": "projetobd",
    "password": "senha123",
    "database": "projetobd",
    "host": "postgres",
    "dialect": "postgres"
  }
}
```

Para executar

```
$ cd bd-backend/
$ make build
$ make migrate-tables
$ make populate-tables
$ make start
```

Frontend - diretório bd-frontend

Não havia necessidade de criar um container para o frontend, uma vez que tudo o que ele usa é instalado pelo yarn e não temos dados para persistir nele, já que ele consulta a API REST que criamos.

Para executar

```
$ cd bd-frontend/
$ yarn install
$ yarn start
```

Acessar aplicação em `localhost:3000/`