Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação

Projeto - Festas - Parte 3

Sistema de Gerenciamento para Assessoria de Eventos

Disciplina: SCC0240 - Bases de Dados
Profª. Dra. Elaine Parros M. de Souza

Turma 2

Aluno	**NUSP**
Felipe Scrochio Custódio	9442688
Gabriel Henrique Campos Scalici	9292970
Danilo Moraes Costa	8921972
Henrique Martins Loschiavo	8936972

Data de Entrega: 24/06/2018

Descrição do Problema

Uma empresa de assessoria de eventos gerencia duas grandes festas tradicionais, uma balada eletrônica e um carnaval com festa à fantasia. As festas acontecem em dias diferentes pré-definidos no ano, no mesmo prédio possuído pela empresa.

Ambos os eventos são organizados de forma que há um preço de entrada no local da festa, onde cada cliente mostra sua documentação, sendo registrado por meio do RG, CPF, data de nascimento (sendo proibido a entrada de menores de 18 anos no local) e recebe uma comanda com um código único e intransferível, onde seus gastos estarão registrados.

A festa de carnaval é uma festa à fantasia. Ao chegar no lugar, o cliente tem a opção de escolher uma fantasia gratuita que será devolvida somente no final da festa, ficando cadastrado em sua comanda o número identificador da fantasia. Cada fantasia possui nome, cor, tamanho e identificador único. Assim, o cliente pode conferir quais fantasias do seu tamanho estão disponíveis, de quais cores e estilos.

O ambiente das festas é dividido em praça de alimentação, bares e pistas com palcos para shows. Por fazerem parte do estabelecimento, em ambas as festas os bares estão presentes. No carnaval, existe uma praça de alimentação com *food trucks*.

A praça de alimentação é composta de vários *food trucks* terceirizados, de diversas culinárias. Para manter controle sobre a terceirização e o uso da praça, os proprietários ficam registrados no sistema da empresa, com CNPJ, nome, RG, CPF, data de nascimento e número de caminhões, sendo que cada proprietário pode ter no máximo 2 caminhões na praça. Os caminhões são cadastrados com placa, nome, proprietário e culinária, podendo ser disponibilizado no site da empresa quais restaurantes estarão disponíveis durante a festa.

Os bares são todos pertencentes ao próprio estabelecimento, que conta com bares em locais diferentes em posições estratégicas. Todos os bares possuem as mesmas bebidas. Os bares são cadastrados por uma identificação única, o número do balcão, que também fica gravado do lado de fora para identificação pelos clientes.

A empresa contrata funcionários para trabalhar nos bares, cadastrando-os por nome, RG, CPF, data de nascimento (os funcionários devem ser maiores de 21 anos) e quanto irá receber. Além do que o funcionário recebe por noite, ele ganha comissão sobre as vendas realizadas. Cada funcionário trabalha em apenas um bar por festa e pode ser consultado quais funcionários estão trabalhando em cada bar.

Todas as compras são adicionadas na comanda, indicando o valor gasto, data com horário, qual ambiente de alimentação (bar ou *food truck*) foi realizada a compra. e qual produto foi consumido. O cliente pode consultar quanto gastou naquela noite utilizando o código de sua comanda nos balcões dos bares.

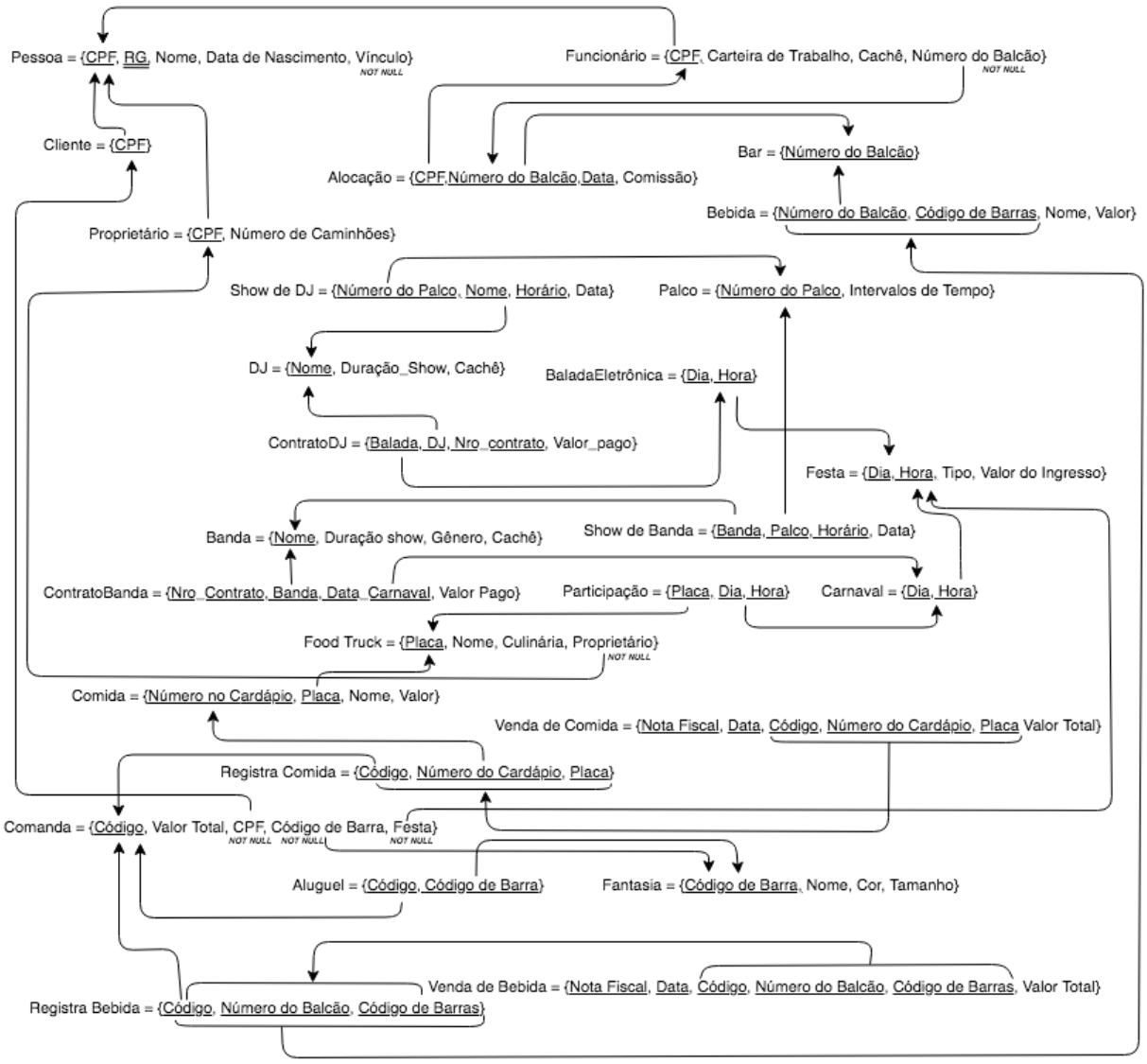
As atrações de ambas as festas são terceirizadas pela empresa, que faz contratos com bandas de diversos estilos para o Carnaval e DJs para a balada eletrônica. Para controle financeiro, as possíveis atrações ficam registradas com seus cachês, que serão lançados no contrato.

A empresa monta no estabelecimento ambientes de música com palcos. Cada palco é identificado pela empresa por um número. No site da empresa, é possível verificar quais atrações irão se apresentar em quais palcos e em quais horários. É possível os usuários fazerem uma consulta no site, para saber quando banda irá tocar no carnaval ou qual DJ irá tocar na balada eletrônica.

Ao sair do estabelecimento deve ser devolvida a comanda e a fantasia, caso seja a festa de carnaval, mediante o pagamento da conta, sendo emitido uma nota fiscal.

Modelo Entidade-Relacionamento (MER)

Modelo Relacional



Decisões de projeto

Pessoa, Cliente, Funcionário e Proprietário: São entidades que fazem parte da estratégia de generalização utilizada na primeira parte do projeto, feitas dessa forma para não se repetissem todos os atributos em várias tabelas diferentes que precisassem de tais campos para sua identificação. Portanto na questão do mapeamento foi criada uma tabela chamada *Pessoa* com todas as informações(*CPF, RG, Nome, Nascimento, Vínculo*) para identificar uma pessoa, seja ela *Cliente, Funcionário* ou *Proprietário* pois ambos possuem os mesmos campos de identificação, sendo possível diferenciá-las pelo seu vínculo com a empresa.

O que foi analisado com mais cautela, foram os atributos específicos de cada entidade que por fim, no mapeamento, foram atribuídos à sua tabela específica, junto com a chave primária *CPF* "herdada" da entidade *Pessoa*, então ficamos com a tabela *Cliente* possuindo apenas a chave *CPF*, *Funcionário* com a chave *CPF* e também os atributos *Carteira de Trabalho* e *Cachê,* e Proprietário também com a chave *CPF,* e atributo *Número de caminhões*.

Vale lembrar que *Cliente, Funcionário e Proprietário* possuem relações diferentes, de forma que são essas tabelas, no mapeamento, que se relacionam, não a tabela *Pessoa* para manter a coerência do banco de dados onde cada um tem o seu papel específico na empresa e nenhum deles tem o mesmo relacionamento.

Balada Eletrônica, Carnaval e Festa: São entidades que fazem parte de uma generalização, que foi assim definida na primeira parte do projeto como forma de resolver o problema de ter duas festas específicas, cada uma com seus relacionamentos, sendo que ambas possuíam os mesmos atributos.

Sendo assim, no mapeamento ficou evidente a otimização de espaço, pois *Balada eletrônica* e *Carnaval* são tabelas que possuem apenas dois atributos (*Dia* e *Hora, a*mbos chave) e se relaciona à entidade festa, que contém todas as informações necessárias (*Dia*, *Hora, Tipo e Valor do Ingresso)* para diferenciar as festas, com atenção principal no conceito de *tipo* em generalizações, pois é na tabela *Festa* que contém todas as informações importantes sobre os eventos, somente quando seu tipo

for *balada* que serão utilizados os valores de *Data* da tabela *Balada Eletrônica*, similar ao que ocorre quando o tipo for *Carnaval*.
De forma que com essa decisão, não seja repetido os mesmos atributos em duas tabelas diferentes, de forma que poupe memória secundária no armazenamento.

Contrato DJ: Tratando-se de uma agregação, que foi assim definida na parte 1 do projeto para que fosse possível armazenar o mesmo "*DJ*" tocando em outras baladas eletrônicas, temos que o mapeamento resultou em uma nova tabela chamada "*Contrato DJ*" possuindo de chave principal o DJ, a Balada e o número de contrato relacionado à eles, também possuindo o "*valor total*" deste contrato, sendo esse um atributo derivado do cachê que o artista cobra para sua apresentação.

Tal mapeamento pode ser considerado trivial, pois não outro jeito de manter a coerência (Um *DJ* tocando em várias baladas e uma balada podendo ter de atração vários *DJ's*), de forma que seja possível pesquisar todas as baladas que um determinado artista tocou, todos que participaram de uma balada, ou quem tocou em qual balada em uma data específica.

Contrato Banda: Como padrão quase sempre para o mapeamento de agregações, o fizemos como *Contrato Banda* uma tabela a parte que relaciona as chaves primárias de *Banda* e *Carnaval*, também a chave *Número contrato* da agregação para que por meio de um número de contrato seja possível relacionar as bandas que foram contratadas para a festa de carnaval.

A agregação possui também um atributo derivado chamado *Valor pago* que foi colocado dessa forma como derivado do cachê que determinada banda cobra por seus shows, informação armazenada na tabela *Banda*, contendo todas as informações necessárias para sua caracterização.

Show de Banda e Show de DJ: São duas tabelas criadas a partir de agregações do modelo relacional, que possuem funcionamento similar.

De maneira análoga à discutida no *Contrato Banda*, temos a necessidade de criar uma tabela a parte para a agregação, de forma que mantenha a possibilidades de armazenamento no banco, isto é, permitir a consulta de quais bandas (ou DJ's) tocaram em quais palcos em determinada data.

Tal tabela possui de chave os campos *Nome da Banda, Número do Palco* e *Horário* (Não optamos por usar data, pois mais bandas podem se apresentar no mesmo dia,

então o que as diferencia é a hora de sua apresentação), pensamento análogo para o show que envolve o DJ, porém com os campos *Nome da Banda, Número do Palco* e *Horário,* de forma que seja mantida a coerência da informação relacionado ao show da banda e DJ, cada um com seus relacionamentos para um tipo de festa específico, podendo ainda obter a informação *Data* derivada da festa pela qual pertence, ou até mesmo consultar o palco para saber quais serão as atrações em cada festa.

Banda e DJ: As duas entidades *Banda e DJ* citada anteriormente tratam-se apenas das informações de uma determinada banda ou DJ que poderá fazer shows apenas na festa de *Carnaval* ou uma *Balada Eletrônica* respectivamente, dessa forma, foram mapeadas para uma tabela, que contenha todas as informações necessárias para caracterizá-las de maneira única, como por exemplo os atributos *Nome, Duração do Show, Gênero* e *Cachê* no caso de *Banda* e *Nome, Duração do show,* e *cachê* no caso do *DJ* .

Foram dois dos mapeamentos mais fáceis de serem feitos no projeto, pois não possuem atributos multivalorados, compostos ou qualquer outro detalhe que deve ser analisado com maior cautela. O ponto principal de análise foi a escolha da chave primária, que foi escolhida como o *Nome* em ambos os casos, onde contamos com a possibilidade dos artistas terem o mesmo nome, porém tomamos como decisão de projeto que isso será tratado em nível de aplicação, adicionando informações adicionais no nome de forma que este se torne um identificador único para cada banda, pois trata-se de uma exceção, que não ocorrerá na maioria das vezes.

Dessa forma estão mantidas a consistência de todas as relações, como por exemplo não existir conflitos de apresentações serem colocadas no mesmo horário.

Palco: Uma entidade simples com apenas dois atributos *Número do palco* e *Intervalos de tempo*, então é de trivial mapeamento, não sendo necessário uma explicação prolongada. Tal entidade foi mapeada em uma *tabela* separada com seus dois atributos e *Número do palco* como chave primária.

Possui dois relacionamentos que podem ser justificados pela necessidade das agregações *Show de Banda* e *Show de DJ* que precisam usar a chave primária de *Palco* para manter consistente os dados sobre as apresentações de cada banda em determinado *Palco*, em determinada *Data* e *Horário.*

Alocação: É uma agregação que relaciona um *Funcionário* trabalhando em um determinado *Bar* em uma determinada *Data*. E como agregação seu mapeamento foi feito com a mesma técnica das demais agregações do trabalho, foi criada uma tabela a parte com chave primária herdando o *CPF* do funcionário e *Número do balcão* que identifica um bar específico e a *Data* específica dessa relação, dessa forma é mantida a coerência, podendo um funcionário trabalhar no mesmo bar em datas diferentes, ou um funcionário trabalhar em outro bar na mesma data, isto porque *Data* é um atributo composto de *Dia* e *Hora*, isto é, pode trabalhar em bares diferentes no mesmo dia, porém em horários diferentes.

Venda de Comida, Venda de bebida, Comida e Bebida: São duas agregações que realizam praticamente as mesmas funções de relacionar uma *Bebida* ou *Comida*, com o tipo de festa, com a *Comanda* de um determinado cliente. E como padrão seguida até agora no projeto, foram utilizadas tabelas separadas para ambas, contendo as chaves primárias das entidades relacionadas e a chave da própria agregação.

Porém temos que tomar cuidado com um ponto muito importante na análise do mapeamento dessas agregações, pois tanto Comida, quanto bebida são entidades fracas que dependem de *Bar* e *Food-Truck* respectivamente, então para manter a consistência das informações que são cadastradas do banco de dados, temos que armazenar nessas tabelas das agregações, a chave primária completa que identifique inequivocamente os produtos.

Porém agora, além de criar as tabelas para as agregações, também criamos, como decisão de projeto, a tabela para o relacionamento *Registra* em ambas, somente para armazenar a relação do produto vendido com a *Comanda*, de forma que *Registra Comida* temos como chave o *Código da Comanda, Número do cardápio e Placa do food-truck* para comida (lembrando mais uma vez que a chave de Comida não é suficiente para identificá-la pois é uma entidade fraca de *Food-Truck,* então precisamos da informação de qual *food-truck* à vendeu dentro da tabela *Comida*), e *Registra Bebida* tem como chave *Código da comanda, Número do Balcão* e *Código de barras* (lembrando que nesse caso só código de barras não é suficiente para identificar inequivocamente uma bebida).

Dessa forma, em relação a *Venda de Comida*, são usadas como chave, a chave primária composta da agregação *Nota Fiscal e Data* (Pois foi analisado que somente nota fiscal poderia haver inconsistências e falhas graves nos dados armazenados) e a chave de *Registra Comida* discutida na alínea acima. Analogamente temos que *Venda*

de Bebida possui como chave *Nota Fiscal e Data* junto com a chave de *Registra Bebida*, também discutida sua consistência na alínea anterior.

Vale a pena citar que a escolha de colocar *Food-Truck* dentro de *Comida* e não vice-versa pois estaríamos economizando espaço de armazenamento.

Fantasia e Aluguel: Fantasia é uma entidade simples, sem necessidade de uma explicação mais detalhada, de forma que a mapeamos para uma tabela separada com sua chave *Código de Barras* e seus atributos simples, *Nome, Cor* e *Tamanho.*

A agregação *Aluguel* está relacionada com a entidade *Fantasia* de modo que a relacione com a *Comanda*, Nesse caso sendo mapeada como uma tabela a parte que contenha apenas dois atributos chave, *Código da Comanda* e *Fantasia*, pois pelo *MER* podemos observar que uma comanda pode conter apenas uma fantasia, satisfazendo a restrição de que um cliente só pode alugar uma fantasia por festa. É possível ver mais detalhes da relação entre *Fantasia* e *Comanda* na explicação do mapeamento da entidade *Comanda* feito mais abaixo.

Food-Truck, Participação: Na nossa análise de requisitos, temos que na festa do tipo *Carnaval* existe uma atração gastronômica, representada aqui pelo *Food-Truck*. Se tivéssemos armazenado dados de *Food-Truck* diretamente em *Carnaval* em uma tentativa de amarrar as duas entidades, teríamos um problema: caso a atração mudasse, teríamos que atualizar toda a tabela de *Carnaval*.

Optamos por uma agregação para resolver este problema, definindo essa relação *Possuir,* permitindo por exemplo manter histórico de quais *Food-trucks* estiveram presentes nas festas de Carnaval, já que temos essa nova tabela com as informações da atração e da festa, podendo, esta atração, ser facilmente alterada.

Outra decisão que tivemos, analisando a participação total de *Food-Truck* com *Proprietário*, foi colocar um campo dentro da tabela *Food-truck* que indicasse quem era o seu *Proprietário*. De forma que não precisássemos armazenar dentro do Proprietário, todos os caminhões que possui, somente a quantidade, pois o gasto de memória seria grande caso houvesse uma grande diferença de quantidade de caminhões entre os donos.

Bar: Uma entidade com apenas apenas atributos um atributo (chave), tendo assim um simples mapeamento. Como ele só se relaciona com entidade fraca (Bebida) e entidade com participação total (Funcionário), portanto, só possui sua própria chave.

Assim, Bebida e Funcionário possuem a chave de Bar (Número do Balcão) como parte da chave composta e chave estrangeira (respectivamente), além da agregação Alocação que também tem a chave como parte da composta.

Comanda: É uma entidade com dois atributos (uma chave), relacionado a várias outras entidades e agregações. Em relação a *Fantasia*, é o caso de várias comandas para uma fantasia, então possui a chave da entidade *Fantasia* (*Código de Barra*) como chave estrangeira. Em relação a *Cliente*, *Comanda* tem participação total, logo tem a chave de *Cliente* (*CPF*) como chave estrangeira.

Optamos como decisão de projeto colocar *CPF* do cliente dentro da tabela *Comanda* pois possui uma relação de um para um, e dentre as formas de mapeamento, optamos dessa maneira para fazer as relações específicas do cliente de forma separada da comanda.

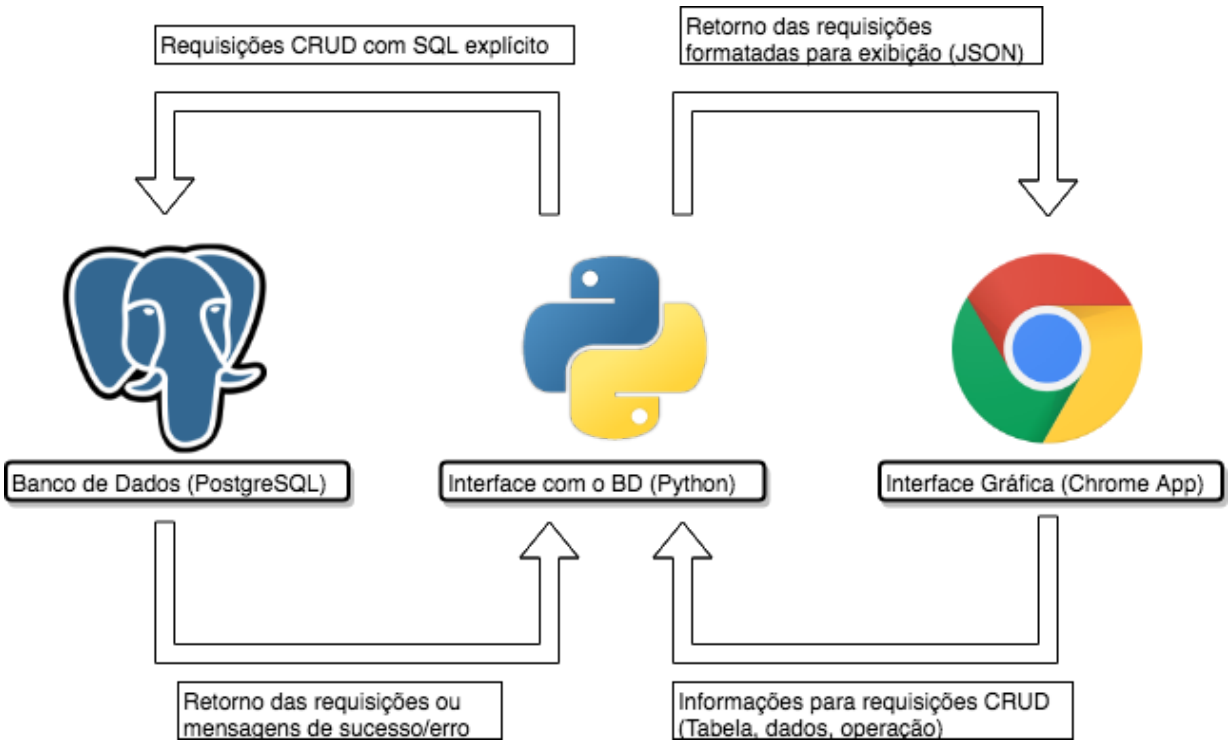
Em relação a *Comida*, é o caso de várias comandas para várias comidas, então é necessário um novo relacional *Registra Comida* contendo ambas as chaves (estrangeiras) de Comanda e Comida. O mesmo pode ser dito em relação a relação com *Bebida* (criando *Registra Bebida*).

Ainda há as agregações, em que todas (*Aluguel, Venda de Comida e Venda de Bebida*) possuem as chaves de seus respectivos atributos relacionados como parte de sua chave composta, além do resto de suas chaves (composta) e atributos (caso possuam).

Implementação

Banco de Dados e Interface

O projeto foi implementado com o banco de dados PostgreSQL, com conexão e requisições em linguagem Python, utilizando o módulo psycopg2. Para a interface gráfica, utilizamos o módulo Python Eel, que sobe um servidor web local e conecta-se a uma página que irá exibir as informações do banco de dados. Esse módulo permite expor funções do Python para o Javascript da página, utilizando a decoração de função *@eel.expose*, e vice-versa, permitindo a separação da aplicação, que trabalha diretamente com o banco de dados, e a interface gráfica, conforme o diagrama a seguir.



Tudo relacionado ao banco de dados é feito no script Python, a interface gráfica apenas envia os dados que o usuário deseja inserir/remover através dos formulários.

Os requisitos do sistema são:

- PostgreSQL
- Python, com pip (gerenciador de pacotes)
- Módulos do Python (listados no arquivo *requirements.txt*)
- Google Chrome ou Chromium. É possível mudar o navegador para outro no código.

Para instalar os módulos Python, utiliza-se o gerenciador de pacotes pip e o arquivo *requirements.txt*.

```
pip install --user -r requirements.txt
```

Esse comando irá instalar os pacotes localmente, no diretório do projeto.

Interface Gráfica

A interface gráfica é uma página web que utiliza HTML/CSS/Javascript. Utilizamos um *template* (*Light Bootstrap Dashboard, de creative-tim*) de painel de controle de administrador com Bootstrap, que facilita o desenvolvimento, diminuindo o tempo gasto na parte gráfica, focando na implementação das requisições e exibição das informações do banco.

Implementação e Execução

O projeto foi implementado e testado nas plataformas Windows, Mac e Linux.
Para execução, primeiro precisamos iniciar o serviço do banco de dados Postgres. Com o Postgres inicializado, é preciso criar um novo servidor, conexão, com quaisquer nomes, e então um banco, com o nome *database*. As informações necessárias estão no arquivo *database.ini*.

Inicialização

Windows	Mac	Linux

Instalamos o **BigSQL PGC** (Postgres Package Manager) via terminal.
 	
Depois instalamos o **PostgreSQL** (versão 10) e executamos:
 	
`pgc install pg10`
 	
Por fim, criamos o servidor e conexão com a shell **psql**.
 	
`psql -U postgres -d mydb`
 	
`create database "database"`
	Instalar o aplicativo **Postgres** baixado diretamente do site do Postgres (não é necessário nenhum comando via terminal, tudo pode ser configurado com a interface gráfica)
	
	Para iniciar o servidor basta clicar no **start** e criar um banco de dados com o nome **database**
	
	Para manipulação dos dados pode ser usado via terminal ou **pageAdmin4** também baixado pelo site do Postgres.
	
	Para fechar a conexão com o servidor, basta clicar no botão **stop**.
		Inicializamos o serviço utilizando *systemctl* ou equivalente.
		
		`sudo systemctl start postgresql`
		
		Depois, criamos o servidor e conexão com a shell **psql**.

database.ini

```
[postgresql]
host=localhost
database=database
user=postgres
password=postgres
```

Utilizamos a ferramenta **pgAdmin4**, que funciona de modo similar ao SQL Developer utilizado nas aulas práticas, para verificar e testar as tabelas e comandos SQL.

Com o banco de dados rodando, podemos executar nossa aplicação. O script ***main.py*** irá realizar a conexão ao banco de dados seguindo o arquivo *database.ini*. A execução é feita pelo comando:

```
python main.py
```

Temos duas telas durante a execução: a interface textual, no terminal, com as saídas mais técnicas para o desenvolvedor, e a interface gráfica, para o usuário leigo, com mensagens de erro e sucesso mais simples.

A conexão é feita pelo módulo psycopg2. Primeiro, geramos uma conexão:

```
connection = psycopg2.connect(**params)
# sempre commitar após um comando,
# assim erros podem ser ignorados, porém exibidos
connection.autocommit = True
```

Essa conexão nos dá um cursor, que é usado para navegar e fazer operações no banco de dados.

```
cursor = connection.cursor()
```

Com essa conexão, podemos realizar nossas consultas e requisições CRUD.

Caso a conexão não seja bem sucedida, uma mensagem de erro é mostrada e a execução é cancelada.

Em caso de sucesso, uma mensagem é exibida e são executados os comandos iniciais no banco de dados. A função

Primeiro é feita a remoção de todas as tabelas, para evitar conflitos entre execuções diferentes da aplicação durante os testes:

```
run_sql('drop.sql')
```

Depois a criação de todas as tabelas:

```
run_sql('initialize.sql')
```

Por fim, todas as inserções:

```
run_sql('insert.sql')
```

Terminados os comandos, a interface gráfica é chamada.
Para inicializar a interface gráfica, chamamos o módulo Eel para a pasta *gui*, onde se encontram os arquivos da página web.

```
# inicializar servidor web local
eel.init('gui')
eel.start('index.html')
```

Com isso, a interface irá abrir em um novo navegador e podemos começar a utilizar a aplicação.

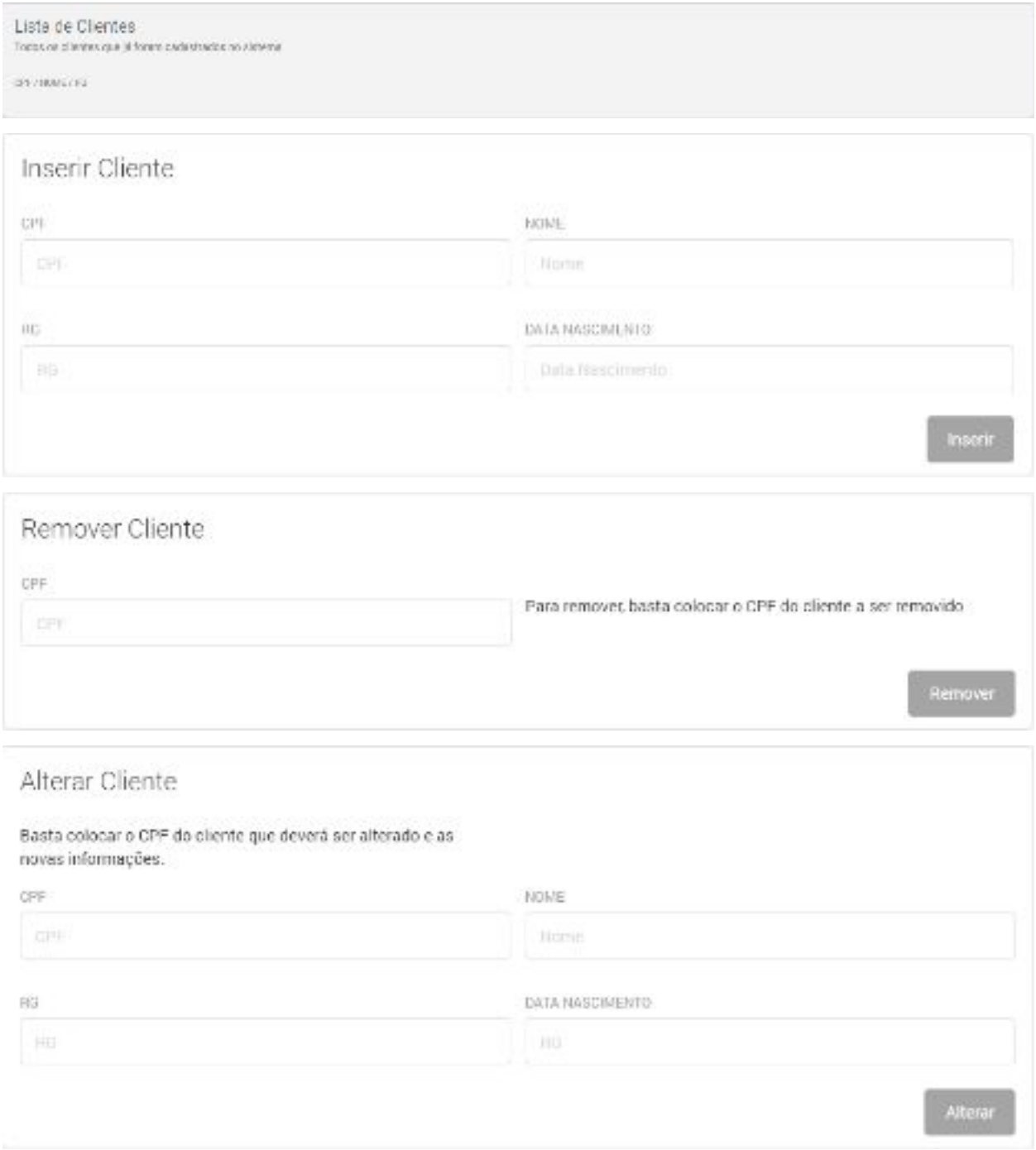
A interface gráfica é dividida nas seguintes telas:

- Home
  - Exibe resultados de consultas mais complexas.

- Listagem e operações CRUD
  - Clientes
  - Funcionários
  - Proprietários
  - Bebidas
  - Comidas
  - Bandas
  - DJs
  - Festa

Cada tela possui os formulários necessários para fazer as operações CRUD, de inserção, remoção e alteração. É possível listar todas as tuplas também. Todos os

exemplos a seguir são com o arquivo **clientes.html**. Os scripts Javascript se encontram no fim da página.



Os botões chamam as respectivas funções Javascript que enviam os conteúdos dos formulários para o script Python, que irá fazer executar a consulta no banco de dados e retornar os valores obtidos. O DOM da página é alterado para exibir os resultados.

Em caso de erro, um alerta é exibido no topo da página.

A implementação de cada requisição CRUD foi feita da seguinte forma no script Python:

Insert

Para a inserção, enviamos para o script em qual tabela desejamos inserir e seus valores.

```
@eel.expose
def insert(table, values):
```

Fazemos a análise dos valores, transformando tudo em uma *string*, concatenando em uma consulta.

```
query = "INSERT INTO " + table + " VALUES (" + values_content + ");"
cursor.execute(query)
```

Executamos o comando de inserção, verificando se algum erro ocorreu, encapsulando-a em um *try/except*.

No Javascript do site, enviamos os valores dos formulários chamando a função insert do script Python, recebemos esse resultado e exibimos o alerta caso o erro ocorra.

```
let cpf = document.getElementById("cpf_add_cli").value;
let rg = document.getElementById("rg_add_cli").value;
let nome = document.getElementById("nome_add_cli").value;
let data_nasc = document.getElementById("dataNasc_add_cli").value;

values = [cpf, rg, nome, data_nasc, "CLIENTE"];
let success = await eel.insert("CLIENTE", values)();

if(success == -1) {
      alert("Erro ao inserir o novo registro, tente novamente");
}
```

Todas as requisições CRUD seguem esse modelo.

Delete

O *Delete* foi implementado seguindo o mesmo modelo do *Insert*, porém com um desafio a mais: é possível ter diversas condições para um *Delete.* Para resolver esse problema, usamos um iterador do Python, concatenando todas as condições com AND, caso existissem.

```
# gerar query com dados do site
    query = "DELETE FROM " + table + " WHERE " + str(columns[0]) +
"=" + "'" + str(values[0]) + "'"
    # caso haja mais de uma condição, adicioná-las
    if (len(columns) > 1):
        for index, content in enumerate(columns):
            if (index > 0):
                query += " AND " + str(columns[index]) + "=" + "'" +
str(values[index]) + "'"
```

Update

O *Update* foi implementado seguindo o mesmo modelo do *Delete*, dessa vez com dois casos em que podemos ter mais de um parâmetro: as alterações e as condições para essas alterações.

```
updates = ""
    for index, content in enumerate(column):
        if (index < len(column) - 1):
            updates += str(column[index]) + "=" + "'" +
str(value[index]) + "'" + ", "
        else:
            updates += str(column[index]) + "=" + "'" +
str(value[index]) + "'"

    # gerar query com dados do site
    query = "UPDATE " + table + " SET " + updates + " WHERE " +
condition_columns[0] + "=" + "'" + condition_values[0] + "'"
```

```
# caso haja mais de uma condição, adicioná-las
    if (len(condition_columns) > 1):
        for index, value in enumerate(condition_columns, start=1):
            query += " AND " + str(condition_columns[index]) + "=" +
"'" + str(condition_values[index]) + "'"
```

Select

No caso do *Select*, o resultado que o banco de dados nos retorna é uma lista com os valores obtidos. Precisamos formatar esses resultados, convertendo-os em string. Enviamos para o Javascript do site como uma lista de strings.

```
# gerar query com dados do site
query = "SELECT " + columns_content + " FROM " + table

cursor.execute(query)
result = cursor.fetchall()

for value in result:
            results.append(str(value))

return results
```

No Javascript do site, recebemos essa lista e alteramos a tabela de resultados, exibindo o que obtivemos do banco ou exibindo alerta de erro.

```
columns = ["CPF", "NOME", "RG"];
let array = await eel.select("CLIENTE", columns)();
// return clientes
let table = document.getElementById("table");

if(array.length > 0){
     for(var i = 0; i < array.length; i++) {
                // create a new row
                var newRow = table.insertRow(table.length);
```

```
                for(var j = 0; j < 1; j++) {
                // create a new cell
                      var cell = newRow.insertCell(j);
                      // add value to the cell
                      cell.innerHTML = array[i];
                }
      }
} else {
      alert("Não há dados armazenados");
}
```

Arquivos .SQL

Para criar todas as tabelas e preenchê-las, implementamos uma função em Python para executar todos os comandos SQL de um arquivo, usando o caractere ; como separador. Como o banco está conectado no modo *auto-commit*, podemos executar todos os comandos e ignorar os que estão com erro, apenas exibindo qual erro foi retornado. Sem o modo *auto-commit,* toda a transação é bloqueada.

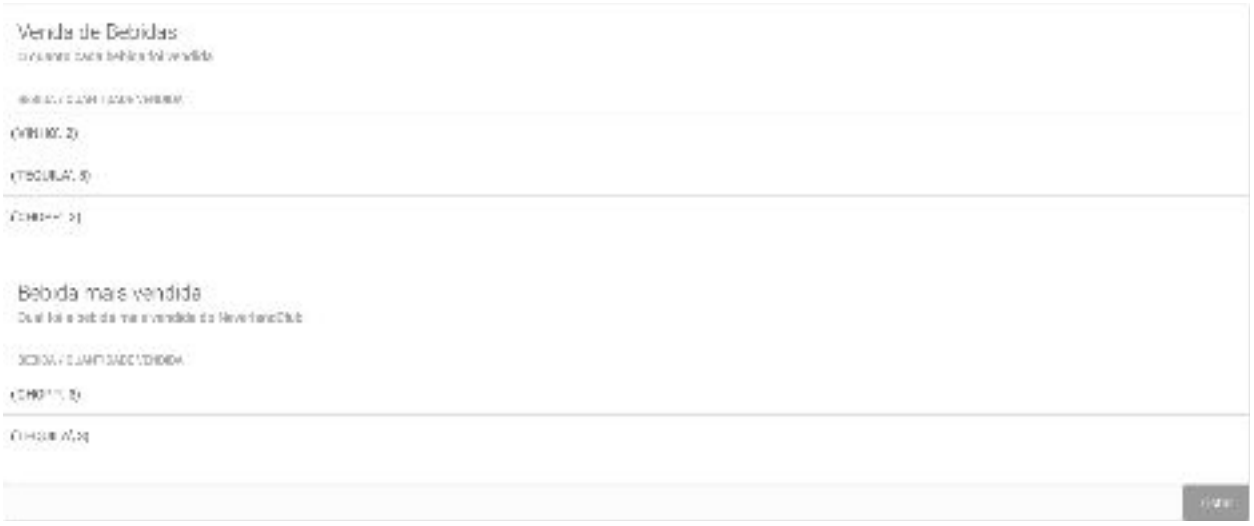
```
# ler arquivo SQL em um único buffer
file = open(filename, 'r')
sql = file.read()
file.close()

commands = sql.split(';')

# executar todos os comandos
for command in commands[:-1]:
    if (len(command) > 0):
        command = command + ';'
        try:
            cursor.execute(command)
        except(Exception, psycopg2.DatabaseError) as error:
            text = colored('ERRO:', 'yellow', attrs=['reverse',
'blink'])
       print('\n' + text + command)
       print('\n' + str(error))
```

Página Inicial

A página inicial possui a exibição dos resultados de algumas das consultas mais complexas. Elas estão na pasta *queries*. O restante das consultas encontra-se no arquivo *select.sql*.



SQL

Criação das tabelas: *initialize.sql*

Gerações de atributos e suas restrições.

Exemplo com chave primária:

```
CREATE TABLE FANTASIA(
    CODIGO_BARRAS   CHAR(13)          NOT NULL,
    NOME            VARCHAR(30),
    COR             VARCHAR(10),
    TAMANHO         CHAR(1),

    CONSTRAINT PK_FANTASIA PRIMARY KEY(CODIGO_BARRAS),
    CONSTRAINT CK_TAMANHO CHECK(UPPER(TAMANHO)
IN('P','M','G','X'))
);
```

Exemplo com chave estrangeira:

```
CREATE TABLE COMANDA(
    CODIGO          INT             NOT NULL,
    CPF_CLIENTE     CHAR(15)        NOT NULL,
    VALOR_TOTAL     REAL            NOT NULL    DEFAULT '0',
    CODIGO_FANTASIA CHAR(13)        DEFAULT '0',
    DIA_FESTA       DATE            NOT NULL,
    HORA_FESTA      TIME            NOT NULL,

    CONSTRAINT PK_COMANDA          PRIMARY KEY(CODIGO),
    CONSTRAINT FK_CPF_CLIENTE     FOREIGN KEY(CPF_CLIENTE)
REFERENCES CLIENTE(CPF)             ON DELETE CASCADE,
    CONSTRAINT FK_CODIGO_FANTASIA FOREIGN KEY(CODIGO_FANTASIA)
REFERENCES FANTASIA(CODIGO_BARRAS)  ON DELETE SET NULL,
    CONSTRAINT FK_FESTA           FOREIGN KEY(DIA_FESTA,
```

```
HORA_FESTA)       REFERENCES FESTA(DIA,HORA)          ON DELETE
CASCADE);
```

Inserções: *insert.sql*

Inserções de tuplas respeitando as restrições.

Exemplo:

```
INSERT INTO PROPRIETARIO
VALUES ('455.111.111.26', '361111126', 'CARLOS CUSTODIO',
'1968-12-17', 'PROPRIETARIO', '1');
```

Consultas: *select.sql*

Consultas aos dados inseridos.

Exemplo de consulta de proprietários e seus respectivos *food trucks*:

```
SELECT
  proprietario.nome AS Proprietario,
  f.nome AS Food_Truck
FROM (
  SELECT
    cpf,
    nome
  FROM
    proprietario) proprietario
LEFT JOIN (
  SELECT
    proprietario,
    nome
  FROM
    food_truck) f
ON
  proprietario.cpf = f.proprietario
```

Exemplo de consulta do nome da bebida mais vendida, caso houver empate,

será mostrado todas as bebidas com o número máximo de vendas:

```
SELECT
  bebida.nome, cod_max.vendas
FROM (
  SELECT
    venda.codigo_barras, venda.vendas
FROM (
  SELECT
    MAX(VENDAS) as vendas
  FROM (
    SELECT
      COUNT(codigo_barras) AS vendas,codigo_barras
    FROM
      venda_bebida
     group by codigo_barras) venda) maximo
  INNER JOIN (
    SELECT
      COUNT(codigo_barras) AS vendas,codigo_barras
    FROM
      venda_bebida
     group by codigo_barras) venda
  ON venda.vendas = maximo.vendas) cod_max
JOIN (
  SELECT
    nome, codigo_barras
  FROM bebida
  ) bebida
ON cod_max.codigo_barras = bebida.codigo_barras
```

No SELECT mais interno, é feita a contagem de vendas por cada código de barra da tabela de vendas de bebida, o SELECT externo a esse extraí o maior número dessa lista e é feito um INNER JOIN deste número com a própria tabela, esta união retornará apenas os códigos de barra associados ao número máximo de vendas. Deste resultado é feito um JOIN com a tabela de bebidas para obter os nomes a partir dos códigos de barra.

Exemplo de consulta das fantasias alugadas com seus respectivos clientes:

```
SELECT
  fantasia.nome,
  aluguel_comanda_cliente.nome
FROM (SELECT nome, codigo_barras
  FROM
    fantasia) fantasia
JOIN (
  SELECT
    aluguel.codigo_fantasia,
    comanda_cliente.nome
  FROM (
    SELECT
      codigo_fantasia,
      codigo
    FROM
      aluguel) aluguel
  JOIN (
    SELECT
      comanda.codigo,
      cliente.nome
    FROM (
      SELECT
        codigo,
        cpf_cliente
      FROM comanda) comanda
    JOIN (SELECT nome, cpf
      FROM
        cliente) cliente
    ON
      cliente.cpf = comanda.cpf_cliente) comanda_cliente
  ON
    comanda_cliente.codigo = aluguel.codigo)
aluguel_comanda_cliente
ON
  fantasia.codigo_barras =
aluguel_comanda_cliente.codigo_fantasia
```

Remoção: *drop.sql*

Remoções das tabelas.

Exemplo:

```
DROP TABLE SHOW_BANDA CASCADE;
```

Conclusão

Aprendemos muito com o projeto durante todo o semestre, sua divisão em três partes ficou bem clara e auxiliou muito na hora de organizar e dividir tarefas, assim como os estudos para as provas, já que cada trabalho abordou o conteúdo da avaliação que teríamos em seguida. Porém, o projeto foi bem longo e tivemos várias dificuldades pois a maioria do grupo estava cursando muitas disciplinas com projetos práticos ao mesmo tempo, alguns até estavam fazendo estágio. No final, ficamos contentes com os resultados e sentimos que conseguimos colocar em prática quase tudo que foi visto em sala de aula.

As maiores dificuldades que tivemos durante a elaboração do projeto foram, nas primeiras etapas entender com clareza como deveríamos organizar os dados em diferentes tabelas, quais dados deveriam ser separados de outros e como fazer suas conexões e restrições. Essa etapa foi a que mais demoramos para começar, pois a disciplina ainda estava no início e nenhum integrante do grupo tinha conhecimento prévio sobre modelagem. Após iniciar os estudos para a primeira prova, a primeira etapa fluiu bem e terminamos o MER.

Na última etapa a maior dificuldade foi a implementação do banco de dados com a integração do cliente web / python. Optamos pela interface gráfica web para facilitar a separação entre a aplicação do usuário e a aplicação do banco de dados, porém isso introduziu o desafio de tratar todos os dados corretamente, enviando dados entre programas rodando em linguagens diferentes: Python e Javascript. Foi uma experiência muito produtiva, pois pudemos aplicar conhecimentos de desenvolvimento web e integração, que são essenciais no mercado atual.

Por mais que a parte 3 do projeto tenha exigido um contato com desenvolvimento web, durante todo o tempo mantivemos o foco no banco de dados, em sua consistência, relações e estrutura de maneira geral, de forma que os mínimos detalhes de SQL e o uso do Postgres tiveram dedicação muito maior pelo grupo. A parte que envolve a página de administração (interface web) foi construída faltando alguns detalhes em alguns formulários, como por exemplo máscaras para campos, checagem de valores nulos, acentos, letras maiúsculas dentre outros, por faltar tempo e experiência em Javascript, principalmente também pelo foco à disciplina de banco de dados. Apesar disso, se o banco de dados retornar um erro, ele é identificado e é exibido um alerta ao usuário.

Sugerimos que para a Parte 3 do projeto, sejam dados exemplos de interfaces a serem criadas e como suas operações são feitas e exibidas. O projeto de exemplo ajudou muito, mas ele só possuía o texto e diagramas das duas primeiras partes.