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

Projeto 3 - Bases de Dados

Fernando Moura Leite Vendrameto - 9875973
Gustavo Sutter Pessurno de Carvalho - 9763193
Rodrigo Geurgas Zavarizz - 9791080
Turma 1

24 de Junho de 2018

Descrição do sistema

Uma empresa trabalha no ramo de festas, mais especificamente com a área de bebidas, sendo responsável por todas as etapas no processo, desde a logística relacionada ao fornecimento até os funcionários que servirão os convidados.

A empresa trabalha com diferentes fornecedores para manter seu estoque com quantidades suficientes para atender suas demandas. Cada fornecedor é determinado por: nome, CNPJ, telefones e dados bancários. Os pedidos para fornecedores são feitos ao final de cada dia, de acordo com o estoque da empresa e a necessidade das festas que a empresa foi contratada no dia. É necessário manter informações desses fornecedores através de cadastros para buscas ou exclusões futuras.

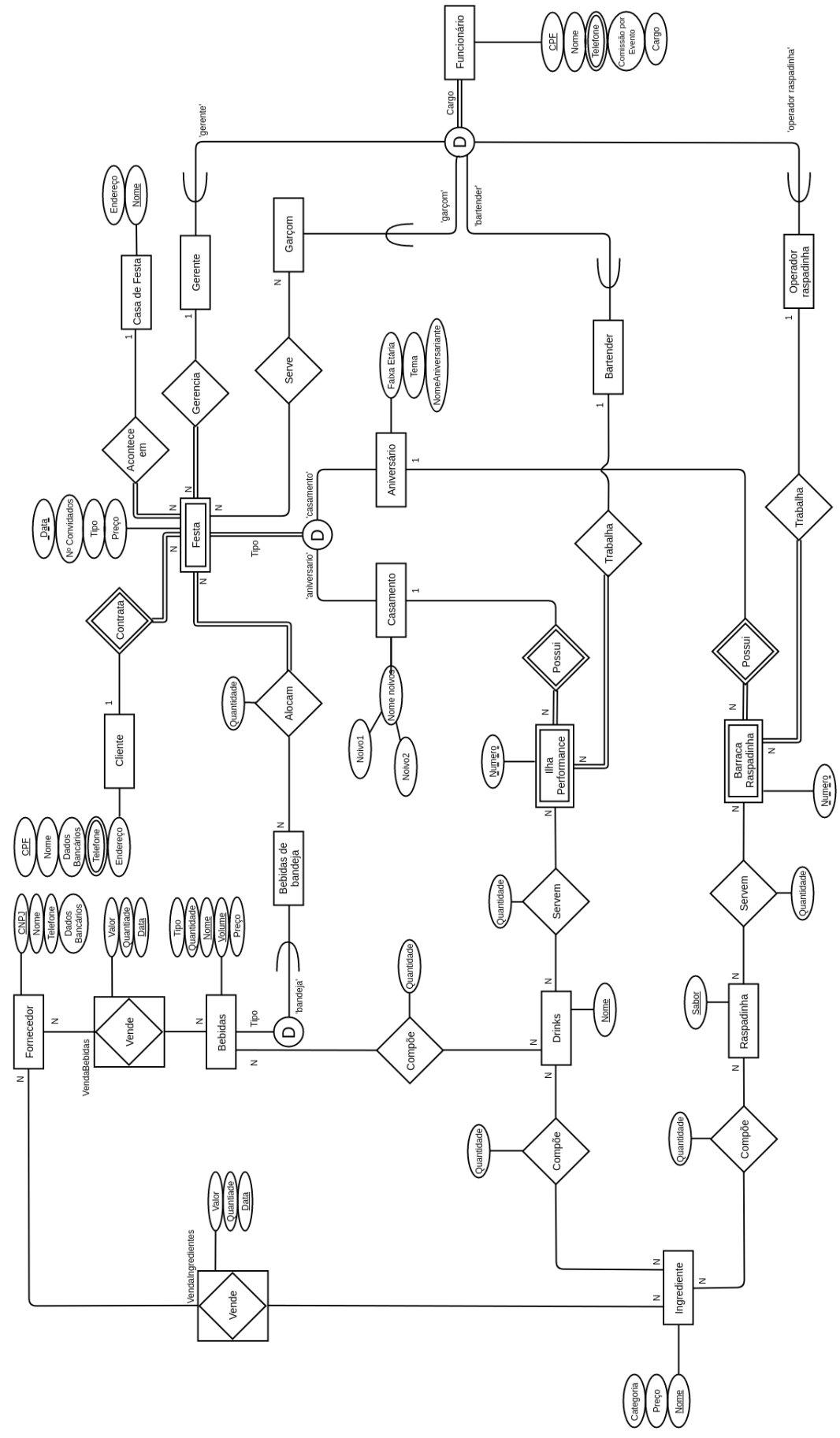
A empresa é contratada para participação em um evento por um cliente, portanto é evidente que cada evento deve possuir um cliente responsável. Este é definido por: nome, CPF, telefone, endereço e dados bancários. Evidentemente é possível que um mesmo cliente esteja envolvido na contratação dos serviços para mais de uma evento, todavia cada evento deve possuir apenas um cliente responsável. O sistema deve ser capaz de incluir os dados do cliente para relacioná-lo a uma festa assim como guardar seus dados, com opção de atualização, para eventos futuros.

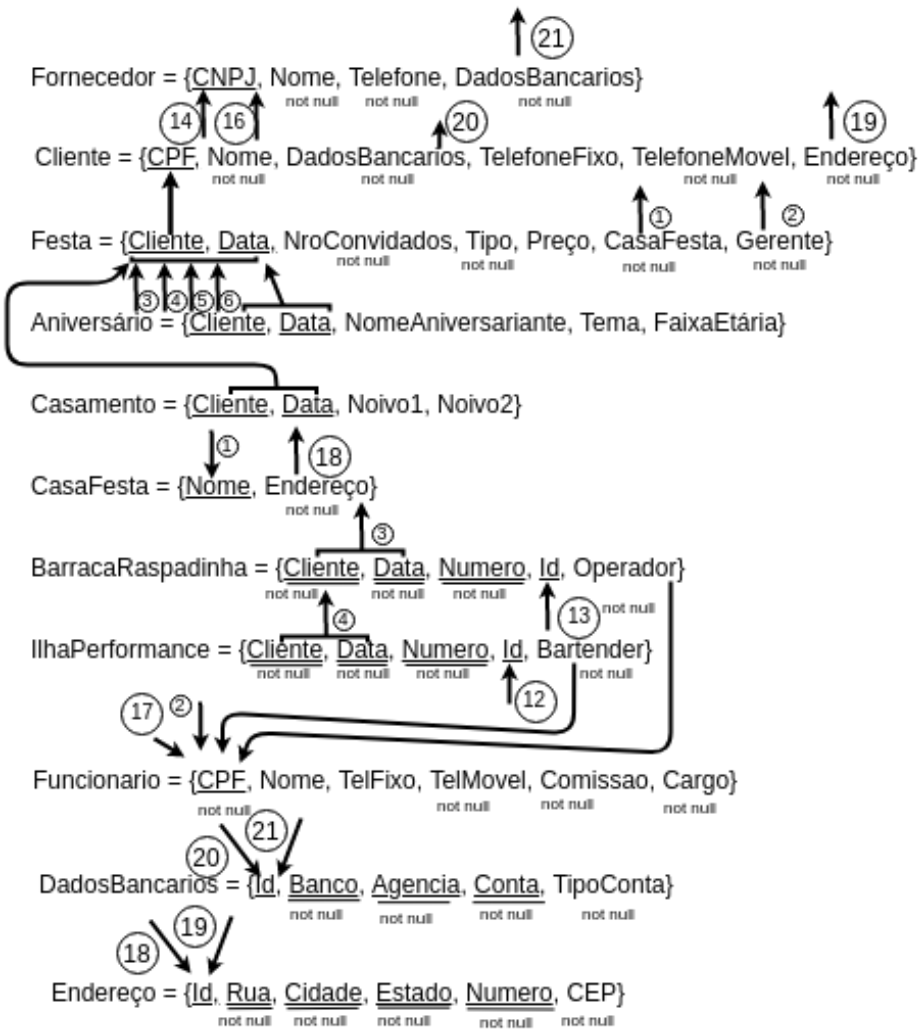
Cada evento possui data, número de convidados, tipo, preço e casa de festas. As casas de festa são armazenadas no sistema, possuindo nome e endereço. As festas atendidas pela empresa variam, podendo ser de casamento ou aniversários infantis, sobre o casamento é armazenado o nome dos noivos e sobre os aniversários o nome do aniversariante, sua faixa etária e o tema da festa.

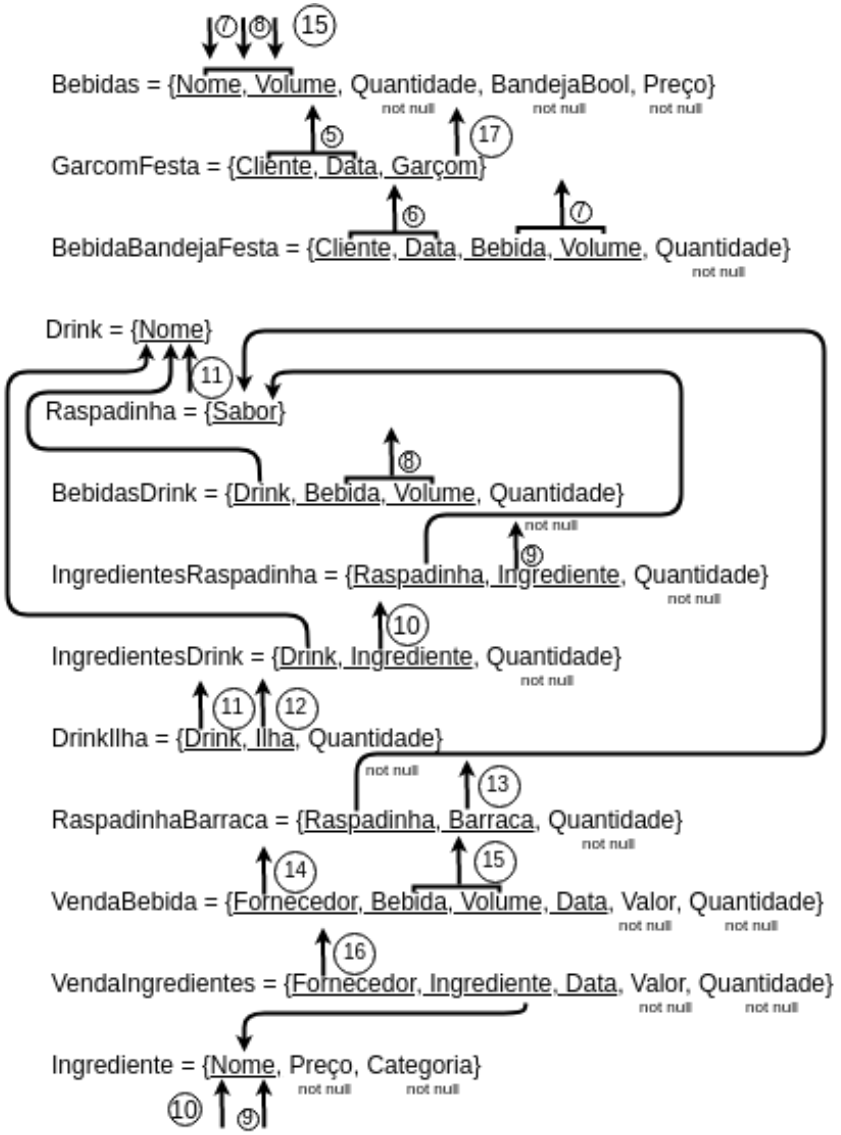
Nas festas de casamento são oferecidas ilhas identificadas por um número que contam com a presença de *bartenders* que realizam performances enquanto servem *drinks* identificados por um nome, compostos por bebidas e ingredientes diversos. Já nas festas infantis são espalhadas barracas onde um profissional específico serve raspadinhas de sabores variados para as crianças, as raspadinhas são feitas na hora da festa e utilizam uma quantidade específica de ingredientes. Sobre os ingredientes utilizados sabe-se o nome, o preço e a quantidade, sendo necessário manter informações sobre eles no sistema.

Também é papel da empresa o envio de seus funcionários para trabalharem no evento. Cada funcionário possui nome, função, CPF, telefone e custo por evento. Existem diferentes funcionários, que desempenham funções específicas: garçons, gerentes, *bartenders*, operador da barraquinha de raspadinha. A demanda por eles é calculada de acordo com as características de cada evento. É importante que cada funcionário trabalhe em apenas um evento por dia. É necessário cadastrar, editar e excluir funcionários para que a empresa mantenha os controle de seus profissionais contratados.

Modelo Entidade-Relacionamento




Fornecedor = {CNPJ, Nome, Telefone, DadosBancarios}
Cliente = {CPF, Nome, DadosBancarios, TelefoneFixo, TelefoneMovel, Endereço}
Festa = {Cliente, Data, NroConvidados, Tipo, Preço, CasaFesta, Gerente}
Aniversário = {Cliente, Data, NomeAniversariante, Tema, FaixaEtária}
Casamento = {Cliente, Data, Noivo1, Noivo2}
CasaFesta = {Nome, Endereço}
BarracaRaspadinha = {Cliente, Data, Numero, Id, Operador}
IlhaPerformance = {Cliente, Data, Numero, Id, Bartender}
Funcionario = {CPF, Nome, TelFixo, TelMovel, Comissao, Cargo}
DadosBancarios = {Id, Banco, Agencia, Conta, TipoConta}
Endereço = {Id, Rua, Cidade, Estado, Numero, CEP}
not null


Bebidas = {Nome, Volume, Quantidade, BandejaBool, Preço}
GarcomFesta = {Cliente, Data, Garçom}
BebidaBandejaFesta = {Cliente, Data, Bebida, Volume, Quantidade}
Drink = {Nome}
Raspadinha = {Sabor}
BebidasDrink = {Drink, Bebida, Volume, Quantidade}
IngredientesRaspadinha = {Raspadinha, Ingrediente, Quantidade}
IngredientesDrink = {Drink, Ingrediente, Quantidade}
DrinkIlha = {Drink, Ilha, Quantidade}
RaspadinhaBarraca = {Raspadinha, Barraca, Quantidade}
VendaBebida = {Fornecedor, Bebida, Volume, Data, Valor, Quantidade}
VendaIngredientes = {Fornecedor, Ingrediente, Data, Valor, Quantidade}
Ingrediente = {Nome, Preço, Categoria}
not null

Justificativas

O mapeamento da tabela de funcionários foi realizado mapeando apenas a entidade genérica, tendo como principal argumento o fato das entidades específicas possuírem exatamente os mesmos atributos. O ponto negativo de tal escolha é que tais entidades específicas participam de relacionamentos diferentes, logo se faz necessário o tratamento em aplicação a fim de garantir que cada a participação no relacionamento está se dando por uma tupla que o cargo condiz com tal participação. Também é importante destacar que tal escolha traz consigo a especialização total demostrada no diagrama de entidade relacionamento.

A entidade especializada bebida de bandeja foi mapeada apenas como um *booleano* na entidade genérica bebida. Este mapeamento leva em conta que não existe participação total na generalização e foi escolhido pela entidade especializada não possuir atributos específicos. O ponto negativo é que a entidade especializada participa de um relacionamento, mas tal garantia não é tão crítica e pode ser realizada em aplicação.

O mapeamento da entidade festa e de suas entidades especializadas, aniversário e casamento, foi realizado criando tabelas para todas as entidades. Isto foi realizado uma vez que todas participam de diferentes relacionamentos e a diferenciação entre aniversário e festa de casamento é critica para o sistema. Com tal escolha foi perdida a garantia de participação total nessa generalização, mas isto não representa grandes problemas para o sistema.

Não foi possível garantir que os funcionários não trabalham em mais de uma festa em um mesmo dia. Isto ocorreu pelo garçom, *bartender* e operador de raspadinhas terem sido mapeados dentro de festa, ilha performance e barraquinha de raspadinha, respectivamente. Já o garçom teve a relação de trabalho mapeado como uma nova tabela, mas pela chave ser composta por cliente, data e o garçom é possível que o mesmo seja inserido em duas festas no mesmo dia (desde que apresentem clientes diferentes).

Foi utilizado um identificador para a ilha performance e para a barraca de raspadinha devido ao fato de sua chave ter ficado muito extensa, o que utilizaria mais memória que o necessário em seus relacionamentos.

Ambas as agregações nas vendas realizadas pelos fornecedor, tanto de ingredientes quanto de bebidas foram mapeadas em uma tabela própria, com fornecedor, ingrediente e data como chaves, já que cada par de fornecedor com bebida ou ingrediente pode gerar mais de uma venda.

Implementação do protótipo

Linguagens, SGDB e pré-requisitos

Para o desenvolvimento do sistema foi escolhida a linguagem de programação **Python** e o sistema gerenciador de banco de dados **Oracle**. Para realizar a comunicação entre ambos foi utilizada a biblioteca *cx_ Oracle* versão 12, que traz para o Python funções capazes de estabelecer conexões e executar comandos no banco de dados. Já o desenvolvimento da interface gráfica foi realizado utilizando o framework Qt, em sua versão 5.

Para rodar o programa é necessário instalar a biblioteca *PyQt5* para *Python 3*, com o seguinte comando (no instalador de pacotes do *Python*):

```
sudo pip3 install PyQt5
```

A biblioteca cx_Oracle é um pouco mais trabalhosa de ser instalada, já que requer as bibliotecas *insantclient-basic* e *instantclient-sdk* que só podem ser obtidas pelo site da Oracle, mediante cadastro. Após a instalação delas, seguindo as instruções do site da Oracle, é possível instalar a biblioteca *cx_ Oracle* usando o seguinte comando:

```
sudo pip3 install cx_oracle
```

Execução

Para rodar o programa é necessário executar, na pasta raiz do projeto, o seguinte comando:

```
python3 mwindow.py
```

Requisitos do sistema

Dado o modelo relacional desenvolvido na fase anterior do projeto foram geradas as tabelas e as inserções iniciais em todas as tabelas do banco. Porém para o desenvolvimento do protótipo apenas um subconjunto de todo a estrutura projetada para o banco de dados.

O sistema se concentra basicamente em torno da entidade festa e as entidades que se relacionam com ela. Por questões de simplicidade apenas as festas de aniversário foram consideradas para o protótipo, porém as festas de casamento possuem funcionamento análogo, portanto tal simplificação não trás grandes perdas em relação a diversidade das informações consideradas.

O protótipo possui suporte às operações de criação, edição, remoção e visualização (com algumas opções de filtragem) das entidades cliente, funcionário, festa, fornecedor, casa de festa e bebidas (que basicamente representa o estoque), apresentando as informações de forma clara e objetiva, separando o conteúdo em diferentes abas a fim de facilitar a visualização.

Durante as etapas anteriores do projeto foi informado que algumas verificações seriam feitas em aplicação, o que foi devidamente endereçado. A verificação de que um funcionário não é destinado a mais de um festa em uma mesma data, a confirmação que a bebida alocada para a festa é uma bebida de bandeja e a confirmação do cargo dos funcionários que irão realizar certas tarefas são exemplos de verificações implementadas.

Estrutura do projeto

Todos os elementos do front-end da aplicação bem como a logica de funcionamento se encontram no arquivo *mwindow.py*, enquanto toda e interação com o banco foi encapsulado na classe criada dbHelper, que se encontra em *dbHelper.py*. Tal escolha foi realizada a fim de aumentar a legibilidade do código da lógica de funcionamento, já que uma vez que a classe que auxilia no uso do banco

de dados possui métodos capazes de executar uma grande comandos SQL de acordo com cada necessidade do projeto.

Todos os comandos SQL pelos métodos da classe dbHelper são explícitos, na maioria das vezes apenas concatenando os valores vindos da aplicação através da interação do usuário. Tais métodos possuem um nível alto de especificidade nas informações que inserem/retornam/editam justamente para que a aplicação principal receba os dados prontos para serem exibidos.

Operações SQL realizadas

Nesta seção são demonstradas algumas operações realizadas pelo programa através da linguagem SQL. Uma vez que alguns desses comandos aparecem em contextos diferentes porém de forma totalmente análoga será demonstrado apenas um exemplo de operação, por questões de clareza e simplicidade.

Inserções

Todas as inserções realizadas no sistema tem o mesmo funcionamento, o programa principal chama um dos métodos de dbHelper para inserir na tabela de interesse, como por exemplo *insertIntoEndereco()*. Dentro da classe auxiliar os valores são entregues a função genérica de inserção *insert()*É importante destacar que caso ocorra algum erro no banco a exceção e tratada e informada ao programa principal. O exemplo abaixo mostra o trecho de código com as funções citadas:

```
#Preprocessa um vetor de valores para concatenar no comando SQL
def _preprocess_values(self, values):
        for i, value in enumerate(values):
                if isinstance(value, str):
                        values[i] = "'" + value + "'"
                elif isinstance(value, int):
                        values[i] = str(value)
                elif isinstance(value, float):
                        values[i] = "%.2f" % (value)
                elif isinstance(value, datetime.datetime) or isinstance(
                    value, datetime.date):
                        values[i] = "TO_DATE('%02d/%02d/%4d', 'DD/MM/YYYY')"
                            % (value.day, value.month, value.year)
                elif value is None:
                        values[i] = "NULL"
        return values
#Executa um comando no banco de dados
def _run_command(self, cmd):
                cursor = self.connection.cursor()
                cursor.execute(cmd)
                cursor.close()
#Insere uma novo tupla na tabela especificada
def insert(self, table, fields, values):
        values = self._preprocess_values(values)
        cmd = 'INSERT INTO ' + table + ' (' + ', '.join(fields) + ') VALUES
            (' + ', '.join(values) + ') '
        self._run_command(cmd)
#Insere um novo enderco no banco de dados.
def insertIntoEndereco(self, values):
        table = 'ENDERECO'
        fields = ['ID', 'RUA', 'CIDADE', 'ESTADO', 'NUMERO', 'CEP']
```

```
        try:
                self.insert(table, fields, values)
        except DatabaseError as e:
                error = getError(str(e))
                return errors[error]
        return INSERT_SUCCESS
```

Atualizações

As operações de atualização são semelhantes as de inserção, é chamado uma função de atualização dos valores de uma tabela específica, *updateEndereco()* por exemplo, que chama a função de atualização genérica *update()*. Assim como na inserção a atualização também trata os erros retornando-os para a interface exibi-los.

```
#Atualiza os dados das tabelas que satisfazem as condicoes de where
def update(self, table, where_fields, where_values, update_fields,
    update_values):
        where_statements = []
        for i, field in enumerate(where_fields):
                where_statements.append(field + ' = ' + where_values[i])
        set_statements = []
        for i, field in enumerate(update_fields):
                set_statements.append(field + ' = ' + update_values[i])
        cmd = 'UPDATE ' + table + ' SET ' + ', '.join(set_statements) + '
            WHERE ( ' + ' AND '.join(where_statements) + ')'
        self._run_command(cmd)

#Atualiza um endereco
def updateEndereco(self, where_values, values):
        values = self._preprocess_values(values)
        where_values = self._preprocess_values(where_values)

        table = 'ENDERECO'
        fields = ['RUA', 'CIDADE', 'ESTADO', 'NUMERO', 'CEP']
        where_fields = ['ID']
        try:
                self.update(table, where_fields, where_values, fields,
                    values)
        except DatabaseError as e:
                error = getError(str(e))
                return errors[error]
        return UPDATE_SUCCESS
```

Remoções

A remoção tem funcionamento similar, o código da aplicação chama a função de remoção da instancia de dbHelper, *delete()*, informando a tabela onde a remoção deve ser realizada os campos que determinam as restrições e, evidentemente as restrições. Uma vez com esses dados as restrições são processadas gerando uma string que remove o que é necessário

```
#Apaga as tuplas de uma tabela dada que cumprem as condicoes fornecidas
def delete(self, table, fields, values):
                values = self._preprocess_values(values)
                where_statements = []
                for i, field in enumerate(fields):
                        where_statements.append('UPPER(' + field + ') = 
                            UPPER(' + values[i] + ')')

                cmd = 'DELETE FROM ' + table + ' WHERE (' + ' AND '.join(
                    where_statements) + ')'
                self._run_command(cmd)
```

Consultas

As consultas são realizadas de forma bem específica, sendo tratadas por diferentes métodos de dbHelper. Portanto serão explicadas separadamente, todavia a função *_run_select()* a seguir é utilizada por todos os métodos que serão descritos.

```
#Executa um comando de select no banco retornando o resultado em uma lista
    de listas
def _run_select(self, cmd):
        cursor = self.connection.cursor()
        cursor.execute(cmd)
        result = []
        for elem in cursor:
                vals = []
                for i, value in enumerate(elem):
                        if isinstance(value, datetime.datetime):
                                vals.append("%02d/%02d/%4d" % (value.day,
                                    value.month, value.year))
                        else:
                                vals.append(str(value))
                result.append(vals)
        cursor.close()
        return result
```

Para obter todas as festas que serão exibidas na tabela para o usuário o programa faz uso da função *getAllAniversarios()* da classe dbHelper, passsando como parâmetros os valores de filtro para as festas que devem ser selecionadas. Como é demonstrado a seguir:

```
#Retorna todas as festas de aniversario que satisfazem restricoes
def getAllAniversarios(self, data_inicio, data_fim, gerente, casa_festa):
        data_inicio, data_fim, processed_gerente, processed_casa_festa =
            self._preprocess_values([data_inicio, data_fim, gerente,
            casa_festa])
        where_constraints = "(F.DATA BETWEEN " + data_inicio + " AND " +
            data_fim + " "
        if(casa_festa != 'Todas'):
                where_constraints += "AND F.CASA_FESTA = " +
                    processed_casa_festa + " "
        if(gerente != "Todos"):
                where_constraints += "AND F.GERENTE = " + processed_gerente
                    + " "
```

```
        where_constraints += ')'
        cmd = "SELECT F.CLIENTE, F.DATA, F.NUMERO_CONVIDADOS, F.PRECO, F.
            GERENTE, F.CASA_FESTA, A.NOME_ANIVERSARIANTE, A.TEMA, A.
            FAIXA_ETARIA, COUNT(GF.GARCOM) \
        FROM FESTA F JOIN ANIVERSARIO A ON A.CLIENTE = F.CLIENTE AND
            A.DATA = F.DATA\
        LEFT JOIN GARCOM_FESTA GF ON (GF.DATA = A.DATA AND GF.
            CLIENTE = A.CLIENTE) \
        WHERE " + where_constraints + "\
        GROUP BY(F.CLIENTE, F.DATA, F.NUMERO_CONVIDADOS, F.PRECO, F.
            GERENTE, F.CASA_FESTA, A.NOME_ANIVERSARIANTE, A.TEMA, A.FAIXA_ETARIA)"
        return self._run_select(cmd)
```

A busca por funcionários é realizada individualmente para cada cargo, de acordo com o que foi solicitado pelo usuário, o que é realizado pelos métodos *getAllGerentes()*, *getAllGarcons()* e *getAllOperadores()* da classe dbHelper. Abaixo está o código para as duas primeiras, uma vez que a última funciona analogamente a de gerente (tendo como única diferença a junção, que é realizada com a tabela barraca raspadinha):

```
#Retorna todos os gerentes cadastrados o sistema
def getAllGerentes(self):
        cmd = "SELECT F.CPF, F.NOME, F.TEL_MOVEL, F.TEL_FIXO, F.COMISSAO,
            MIN(FE.DATA) AS DATA_PROXIMA_FESTA\
        FROM FUNCIONARIO F LEFT JOIN FESTA FE ON FE.GERENTE = F.CPF
            AND FE.DATA > SYSDATE AND FE.TIPO = 'A' \
        WHERE UPPER(F.CARGO) = 'GERENTE' \
        GROUP BY(F.CPF, F.NOME, F.TEL_MOVEL, F.TEL_FIXO, F.COMISSAO)
            "
        return self._run_select(cmd)

#Retorna todos os garcons cadastrados o sistema
def getAllGarcons(self):
        cmd = "SELECT F.CPF, F.NOME, F.TEL_MOVEL, F.TEL_FIXO, F.COMISSAO,
            MIN(GF.DATA) AS DATA_PROXIMA_FESTA\
        FROM FUNCIONARIO F LEFT JOIN GARCOM_FESTA GF ON GF.GARCOM =
            F.CPF\
        LEFT JOIN FESTA FE ON GF.CLIENTE = FE.CLIENTE AND GF.DATA =
            FE.DATA AND FE.DATA > SYSDATE AND FE.TIPO = 'A' \
        WHERE UPPER(F.CARGO) = 'GARCOM' \
        GROUP BY(F.CPF, F.NOME, F.TEL_MOVEL, F.TEL_FIXO, F.COMISSAO
            )"
        return self._run_select(cmd)
```

O programa realiza uma chamada à função *getAllClientes()* para obter os dados cadastrais de cada cliente bem como a data de sua próxima festa cadastrada no sistema. Tal operação executa o comando a seguir:

```
#Retorna todos os clientes cadastrados o sistema
def getAllClientes(self):
        cmd = "SELECT C.CPF, C.NOME, C.TEL_FIXO, C.TEL_MOVEL, D.BANCO, D.
            AGENCIA, D.CONTA, D.TIPO_CONTA, COUNT(F.CLIENTE) AS NUMERO_FESTAS
            \
        FROM CLIENTE C JOIN DADOS_BANCARIOS D ON C.DADOS_BANCARIOS =
            D.ID\
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣LEFT␣JOIN␣FESTA␣F␣ON␣C.CPF␣=␣F.CLIENTE\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣GROUP␣BY(C.CPF,␣C.NOME,␣C.TEL_FIXO,␣C.TEL_MOVEL,␣D.BANCO,␣D.
    AGENCIA,␣D.CONTA,␣D.TIPO_CONTA)"
        return self._run_select(cmd)
```

Um processo análogo também é utilizada para a obtenção de todas as cases de festa para a exibição da mesma para o usuário, tendo como única diferença a junção com a tabela de endereço.

No momento da criação de uma festa, quando estão sendo escolhidos os garçons que irão trabalhar no evento, é necessário garantir duas coisas: que o funcionário em questão é um garçom e que ele está livre no dia da festa. Para isso apenas os funcionários que cumprem as restrições descritas são exibidos. Isso é obtido pela função a seguir:

```
#Retorna todos os garcons livres na data indicada
def getGarconsLivres(self, data):
        data = self._preprocess_values([data])[0]
        cmd = "SELECT␣F.NOME,␣F.CPF,␣F.COMISSAO␣FROM␣FUNCIONARIO␣F␣WHERE␣
            UPPER(F.CARGO)␣=␣'GARCOM'␣AND\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣F.CPF␣NOT␣IN(\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣GF.GARCOM␣FROM␣GARCOM_FESTA␣GF␣WHERE␣GF.DATA␣=␣" +
    data + ")"
        return self._run_select(cmd)
```

Um processo semelhante também é realizado para a escolha do gerente da festa e dos operadores de raspadinha. Porém uma vez que os relacionamentos para estes cargos são mapeados de forma diferente os comandos também são diferentes. As funções do dbHelper chamadas são *getOperadoresLivres(data)* e *getGerentesLivres(data)*, que funcionam da seguinte forma:

```
#Retorna todos os operadores de raspadinha livres na data indicada
def getOperadoresLivres(self, data):
        data = self._preprocess_values([data])[0]
        cmd = "SELECT␣F.NOME,␣F.CPF,␣F.COMISSAO␣FROM␣FUNCIONARIO␣F␣WHERE␣
            UPPER(F.CARGO)␣=␣'OPERADOR'␣AND\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣F.CPF␣NOT␣IN(\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣BR.OPERADOR␣FROM␣BARRACA_RASPADINHA␣BR␣WHERE␣BR.DATA␣
    =␣" + data + ")"
        return self._run_select(cmd)

#Retorna todos os gerentes livres a data indicada
def getGerentesLivres(self, data):
        data = self._preprocess_values([data])[0]
        cmd = "SELECT␣F.NOME,␣F.CPF,␣F.COMISSAO␣FROM␣FUNCIONARIO␣F␣WHERE␣
            UPPER(F.CARGO)␣=␣'GERENTE'␣AND\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣F.CPF␣NOT␣IN(\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣FE.GERENTE␣FROM␣FESTA␣FE␣WHERE␣FE.DATA␣=␣" + data + "
    )"
        return self._run_select(cmd)
```

Além destas consultas anteriormente descritas existem várias outras funções no dbHelper que realizam consultas no banco de dados, mas em sua grande maioria extremamente simples. Tais comandos são utilizados para obter informações sobre uma instância especifica de uma tabela, como por exemplo em *getDadosBancariosFornecedor()*, *getEnderecoCasaFesta()*, *getNomeCasasFesta()*, dentre outros métodos.

Conclusões

O desenvolvimento do projeto como um todo foi positivo para a revisão e fixação dos conteúdos vistos em sala de aula. Uma vez que o projeto foi desenvolvido durante todo o semestre ficou evidente como cada novo recurso aprendido em aula impactou o projeto, fato que muita vezes fez com que erros cometidos em uma fase anterior ficassem claros na etapa seguinte. E não somente erros, principalmente entre as etapas 2 e 3 ficou claro como diversas escolhas de projeto impactam a construção do sistema real.

As maiores dificuldades do projeto foram em relação ao uso de ferramentas, dentre elas a escolha e o aprendizado da biblioteca que se relaciona com o banco de dados e o uso do framework para criação de interfaces gráficas. Além disso a gerência do projeto em grupo também representou um desafio, porém não impossibilitou a realização de maneira alguma. Como é possível perceber as dificuldades enfrentadas não são advindas do conteúdo da disciplina, mas sim de conceitos mais amplos de desenvolvimento de software.