UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

Projeto — Assessoria de Eventos (Parte 3)

Festas Gastronômicas e Coquetéis de Lançamento

Disciplina: Bases de Dados (SCC0240) — Turma 2
Docente: Elaine Parros M. de Sousa

Alunos:

David Souza Rodrigues (nº USP 4461180)
Gabriel Toschi de Oliveira (nº USP 9763039)
Marcos Wendell Souza de Oliveira Santos (nº USP 9791351)

São Carlos, 24 de junho de 2018

DESCRIÇÃO DO PROBLEMA

A FF Assessoria de Eventos está se estabelecendo no mercado de assessoria de festas, com foco em feiras gastronômicas e coquetéis de lançamento. Seu trabalho envolve promover festas modernas com diversas opções de alimentação e bebidas para os frequentadores.

A empresa busca realizar parcerias a fim de estimular os participantes a terem novas experiências gastronômicas e culturais por meio de festas que atraem públicos diversos, sob a marca **Fest Food**. Além disso, a FF Assessoria de Eventos também auxilia outras empresas e instituições a promoverem **coquetéis de lançamento** para novos produtos ou serviços, atuando nas áreas de logística, infraestrutura e alimentação.

As próximas subseções descrevem, em detalhes, características comuns a ambos os eventos e os formatos padrões de festas que a FF organiza e assessora: Fest Food e coquetéis de lançamento. Por fim, na última seção são apresentadas as funcionalidades principais do sistema.

Definições comuns

Uma festa, independente do formato, necessita de um **local** que comporte o número esperado de participantes para ser realizada. Cada local é identificado pela combinação única do nome e da cidade onde ele se localiza. Um local também contém o resto do seu endereço (rua, número e estado), capacidade máxima de frequentadores, se o local é aberto ou fechado, presença de palcos para apresentações e valor da diária de locação.

Durante a organização de uma festa, há várias **empresas** que participam dos processos relacionados a ela. Uma empresa é definida, de forma genérica, pelo seu nome fantasia, sua razão social (única), seu CNPJ (único), seu tipo (fornecedora e/ou contratante), seu endereço — formado pela combinação do nome da rua, do número, da cidade e do estado —, e um ou mais contatos, formados por nome, e-mail e telefone.

Em toda festa, há uma ou mais **empresas fornecedoras** de alimentos e bebidas aos participantes. Essas empresas são parceiras da FF e sua logística de participação na festa depende do tipo de evento e será explicitada nas próximas seções. Além da definição geral de empresa, as fornecedoras também são definidas por uma ou mais categorias de fornecimento dos produtos que podem suprir. Uma **categoria de fornecimento** é definida por um nome único de referência e uma descrição sobre os alimentos e bebidas que ela representa. Cada categoria de fornecimento relaciona-se com uma empresa, sendo que uma empresa fornecedora pode ter uma ou mais categorias de fornecimento; nesta relação, é explicitada a faixa de preço de uma empresa fornecedora

comparada às demais em relação a uma mesma categoria de fornecimento, que vai de 1 a 5 (1 para “muito barato” e 5 para “muito caro”).

Toda festa também tem, obrigatoriamente, uma **empresa contratante**, que contrata a FF para organizar a logística do seu evento. Uma empresa pode acumular ambas as funções, porém, ela deve ter, pelo menos, uma dessas alcunhas.

A FF Assessoria de Eventos também é responsável por contratar **funcionários** que trabalham durante as festas. Um funcionário é definido pelo seu nome, seu CPF (único), seu RG, seu endereço — formado pela combinação do nome da rua, do número, da cidade e do estado —, seu e-mail, um ou mais telefones para contato, pela sua função e por seu valor por hora trabalhada.

Por fim, pode-se definir **festa** como um evento realizado pela FF Assessoria de Eventos. Cada festa tem, obrigatoriamente, uma empresa contratante relacionada e é identificada pela junção única do CNPJ da empresa contratante e da data de realização do evento. É realizada apenas uma festa, pela mesma empresa contratante, no mesmo dia. Uma festa também possui, obrigatoriamente, um nome, um horário de início, um tempo de duração.

Coquetel de Lançamento

Um **coquetel de lançamento** consiste em uma festa encomendada por uma empresa contratante, que deseja promover o lançamento de um produto ou serviço por meio de um evento social com aperitivos e bebidas. Esse tipo de festa apresenta um público restrito, selecionado pela empresa contratante e controlado por meio de uma lista de convidados.

Um **convidado** é identificado pelo seu e-mail (único); além disso, armazenam-se os dados do seu nome e telefone. Além disso, em alguns coquetéis, **brindes** exclusivos (para cada festa) são oferecidos a critério da empresa contratante, que também define a quantidade a ser distribuída. Um brinde é definido por seu nome específico (único) e descrição.

Em todo coquetel de lançamento, empresas fornecedoras são contratadas para fornecer os aperitivos e bebidas servidos durante a festa. Neste processo de contratação, são armazenados o preço total a ser pago para a empresa fornecedora e uma relação das comidas e bebidas a serem fornecidas, com a finalidade da produção de cardápios e afins.

Para os funcionários que trabalham em um coquetel, é definido, entre ambas as partes, a quantidade de horas que o funcionário trabalhará nessa festa e, a partir desta informação, é calculado o valor a ser pago ao funcionário.

Um coquetel também apresenta um orçamento, que é calculado pela soma do preço de cada fornecimento de alimentos e bebidas, pela diária da locação do local e o valor a ser pago para todos os funcionários contratados.

Fest Food

Uma **Fest Food** trata-se de um outro formato de festa que a FF assessora, promovida por uma empresa contratante e que reúne empresas vinculadas ao ramo gastronômico em um espaço de entretenimento e diversidade.

As Fest Food são eventos abertos ao público com entrada realizada por meio da compra de ingressos. Um **ingresso** é composto por seu número de série e apresenta o CPF do seu comprador, quando é adquirido. Um ingresso deve, obrigatoriamente, estar vinculado a uma Fest Food. A combinação entre o número de série de um ingresso e o identificador da Fest Food a qual ele pertence é única e identifica um ingresso em específico. Todos os ingressos vendidos para uma mesma Fest Food tem o mesmo preço de venda, definido nas informações da própria festa.

Como descrito na definição geral de festa, uma Fest Food acontece em um local específico. Entretanto, nesse tipo de festa, a relação entre uma Fest Food e o local onde ela acontece dá origem a uma **locação**. Uma locação é identificada pela combinação única entre os identificadores de uma Fest Food e de seu local de realização.

Uma locação é seccionada em **lotes** com dimensões predeterminadas e equivalentes, distribuídos de modo a facilitar a circulação dos participantes. Cada lote tem suas dimensões (comprimento e largura), um número de identificação dentro de sua locação e um preço a ser pago para locar o lote. As empresas fornecedoras podem locar os lotes e, assim, garantirem sua participação no evento, podendo divulgar e vender os seus produtos.

O **segurança** é uma função especial de um funcionário. Além dele trabalhar em coquetéis de lançamento, os seguranças são os únicos funcionários que podem trabalhar também em uma Fest Food. Caso isso ocorra, o cálculo de seu salário é feito de forma semelhante ao seu trabalho em um coquetel (o produto entre a quantidade de horas trabalhadas e o seu valor por hora).

Funcionalidades

- Inserção, remoção e atualização das entidades: empresa; local; lote; funcionário; brinde; ingresso; categoria de fornecimento; Fest Food; Coquetel;

- Consultas:
    - Por meio da data, procurar todos os locais disponíveis para a realização de uma festa, seja ela uma Fest Food ou Coquetel;
    - Ao consultar um coquetel, mostrar o preço do orçamento e todos os dados necessários para a construção do mesmo;
    - Para cada categoria de fornecimento, informar as empresas que trabalham com esse tipo de produto e suas faixas de preço;
    - Dada uma cidade, informar os funcionários disponíveis nessa localidade;
    - Informar todas as festas que ocorrem em uma mesma data;
    - Para cada coquetel, informar os convidados relacionados a essa festa;
    - Para uma Fest Food, informar todos os ingressos vendidos;
    - Para um coquetel, informar, se houver, todos os brindes a serem entregues aos convidados durante o evento e, caso não haja, exibir uma mensagem informando que aquele coquetel não apresentará brindes;
    - Para uma locação, informar todos os lotes, o estado deles (adquiridos ou disponíveis) e, caso possível, as empresas fornecedoras que os adquiriram;
    - Dado um coquetel, informar as empresas que fornecerão alimentos e bebidas e quais produtos cada uma delas fornecerá.

Modelo Relacional

Especificação dos domínios

- **Cidade**: conjunto de todos os nomes possíveis para cidades no Brasil — string de 40 caracteres
- **CNPJ**: conjunto dos CNPJs possíveis para empresas no Brasil — string de 14 dígitos
- **CPF**: conjunto de todos os CPFs válidos para brasileiros — string de 11 dígitos
- **Data de Festa**: conjunto das datas possíveis para festas — data no formato DD/MM/AAAA
- **Descrição de Brinde**: conjunto de todos as possíveis descrições que descrevem o que é um brinde — string de 60 caracteres
- **Descrição de Categoria de Fornecimento**: conjunto das descrições possíveis para categorias de fornecimento de alimentos e bebidas — string de 140 caracteres
- **Dimensão de Lote**: conjunto de todos os valores possíveis para uma dimensão unidimensional de um lote — número real positivo
- **Duração de Festa**: conjunto de todas as quantidades de tempo que uma festa pode durar — timestamp no formato HH:MM entre 00:00 e 23:59
- **E-mail**: conjunto de todos os endereços de e-mail possíveis — string de 60 caracteres
- **Endereço de Empresa**: conjunto dos endereços possíveis para empresas no Brasil — string de 120 caracteres

- **Estado**: conjunto de todos os nomes possíveis para estados (Unidades Federativas) no Brasil — string de 19 caracteres
- **Existência de Local Aberto**: conjunto das indicações se um local é aberto ou não — booleano
- **Faixa de Preço de Fornecimento de Produtos**: conjunto de todos os valores possíveis para indicar a faixa de preço dos produtos fornecidos por uma empresa em uma certa categoria de fornecimento — inteiro que assume apenas valores do conjunto {1, 2, 3, 4, 5}
- **Função de Funcionário**: conjunto de todas as funções possíveis para um funcionário — string de 12 caracteres
- **Hora de Início de Festa**: conjunto de todos os horários possíveis para festas — timestamp no formato HH:MM entre 00:00 e 23:59
- **Identificador de Festa**: conjunto de todos os números utilizados para identificar unicamente uma festa no sistema — inteiro
- **Identificador de Locação**: conjunto de todos os números utilizados para identificar unicamente uma locação no sistema — inteiro
- **Identificador de Lote**: conjunto de todos os números utilizados para identificar unicamente uma locação no sistema — inteiro
- **Nome de Brinde**: conjunto de todos os nomes possíveis para um brinde — string de 30 caracteres
- **Nome de Categoria de Fornecimento**: conjunto dos nomes possíveis para categorias de fornecimento de alimentos e bebidas — string de 20 caracteres
- **Nome de Festa**: conjunto de todos os nomes possíveis para festas — string de 60 caracteres
- **Nome de Local**: conjunto de todos os nomes possíveis para locais — string de 60 caracteres
- **Nome de Pessoa**: conjunto de todos os nomes possíveis para pessoas — string de 60 caracteres
- **Nome de Produto**: conjunto de todos os nomes possíveis para um produto fornecido a um coquetel — string de 30 caracteres
- **Nome Fantasia**: conjunto dos nomes fantasia possíveis para empresas no Brasil — string de 60 caracteres
- **Numeração de Ingresso**: conjunto de todas as numerações possíveis para ingressos — inteiro
- **Numeração de Rua**: conjunto de todas as numerações possíveis para endereços no Brasil — inteiro
- **Número de Frequentadores**: conjunto de todos os números máximos de frequentadores para locais — inteiro
- **Preço de Fornecimento de Produtos**: conjunto de todos os preços que podem ser pagos a uma empresa para fornecer produtos a um coquetel — número real positivo
- **Preço de Ingresso**: conjunto de todos os preços que podem ser pagos para um único ingresso de uma Fest Food — número real positivo

- **Preço de Lote**: conjunto de todos os preços que podem ser pagos por uma fornecedora para alugar um lote em uma Fest Food — número real positivo
- **Preço de Orçamento**: conjunto de todos os preços que podem ser calculados para um orçamento de um coquetel — número real positivo
- **Preço Pago a Funcionário por Hora**: conjunto de todos os valores possíveis para o pagamento (por hora) a um funcionário por seu trabalho em uma festa — número real positivo
- **Quantidade de Fornecimento de Produtos**: conjunto de todas as quantidades possíveis de unidades de um mesmo produto a ser fornecido a um coquetel — inteiro maior que 0
- **Quantidade de Horas Trabalhadas por Funcionário**: conjunto de todas as quantidades possíveis de horas trabalhadadas por um funcionário em uma festa — número real positivo
- **Razão Social**: conjunto das razões sociais possíveis para empresas no Brasil — string de 120 caracteres
- **RG**: conjunto de todos os RGs válidos para brasileiros — string de 15 caracteres
- **Rua**: conjunto de todos os nomes possíveis para ruas no Brasil — string de 40 caracteres
- **Telefone**: conjunto de todos os números de telefone possíveis no Brasil — string de 11 dígitos
- **Tipo de Festa**: conjuntos de todos os tipos que uma festa pode ser — string de 8 caracteres que assume apenas os valores “FoodFest” ou “Coquetel”
- **Valor de Diária**: conjunto dos valores possíveis (em reais) a serem pagos por um serviço — número real entre 0,00 e 9.999.999,99
- **Valor Pago a Funcionário**: conjunto de todos os valores possíveis a serem pagos para um funcionário pelo seu trabalho em uma festa — número real positivo

- **Lote**
  - Dom(Fornecedora) = CNPJ
  - Dom(Número) = Identificador de Lote
  - Dom(Locação) = Identificador de Locação
  - Dom(Preço) = Preço de Lote
  - Dom(Largura) = Dimensão de Lote
  - Dom(Comprimento) = Dimensão de Lote
- **ContatoEmpresa**
  - Dom(Empresa) = CNPJ
  - Dom(Email) = E-mail
  - Dom(Nome) = Nome de Pessoa
  - Dom(Telefone) = Telefone
- **Empresa**
  - Dom(CNPJ) = CNPJ
  - Dom(NomeFantasia) = Nome Fantasia
  - Dom(RazãoSocial) = Razão Social
  - Dom(Endereço) = Endereço de Empresa
- **Fornecedora**
  - Dom(CNPJ) = CNPJ
- **Contratante**
  - Dom(CNPJ) = CNPJ

- **FornecimentoCoquetel**
  - Dom(Fornecedora) = CNPJ
  - Dom(Coquetel) = Identificador de Festa
  - Dom(Preço) = Preço de Fornecimento de Produtos
- **ProdutosFornecidos**
  - Dom(Fornecedora) = CNPJ
  - Dom(Coquetel) = Identificador de Festa
  - Dom(Nome) = Nome de Produto
  - Dom(Quantidade) = Quantidade de Fornecimento de Produtos
- **AtribuiçãoCategoria**
  - Dom(Fornecedora) = CNPJ
  - Dom(Categoria) = Nome de Categoria de Fornecimento
  - Dom(FaixaPreço) = Faixa de Preço de Fornecimento de Produtos
- **CategoriaFornecimento**
  - Dom(NomeRef) = Nome de Categoria de Fornecimento
  - Dom(Descrição) = Descrição de Categoria de Fornecimento
- **Locação**
  - Dom(Id) = Identificador de Locação
  - Dom(FestFood) = Identificador de Festa
  - Dom(Nome) = Nome de Local
  - Dom(Cidade) = Cidade
- **Local**
  - Dom(Nome) = Nome de Local
  - Dom(Cidade) = Cidade
  - Dom(Estado) = Estado
  - Dom(Rua) = Rua
  - Dom(Número) = Numeração de Rua
  - Dom(MaxFrequentadores) = Número de Frequentadores
  - Dom(PossuiAbertura) = Existência de Local Aberto
  - Dom(DiáriaLocação) = Valor de Diária
- **Ingresso**
  - Dom(FestFood) = Identificador de Festa
  - Dom(Número) = Numeração de Ingresso
  - Dom(CPFComprador) = CPF
- **FestFood**
  - Dom(Festa) = Identificador de Festa
  - Dom(PreçoIngresso) = Preço de Ingresso
- **Festa**
  - Dom(Contratante) = CNPJ
  - Dom(Data) = Data de Festa
  - Dom(Festa) = Nome de Festa
  - Dom(HoraInício) = Hora de Início de Festa
  - Dom(Duração) = Duração de Festa
  - Dom(TipoFesta) = Tipo de Festa
  - Dom(ID) = Identificador de Festa
- **Coquetel**
  - Dom(Festa) = Identificador de Festa
  - Dom(Orçamento) = Preço de Orçamento
  - Dom(Local) = Nome de Local
  - Dom(Cidade) = Cidade

- **Brinde**
    - Dom(Coquetel) = Identificador de Festa
    - Dom(Nome) = Nome de Brinde
    - Dom(Descrição) = Descrição de Brinde
- **Convite**
    - Dom(Convidado) = E-mail
    - Dom(Coquetel) = Identificador de Festa
- **Convidado**
    - Dom(E-mail) = E-mail
    - Dom(Nome) = Nome de Pessoa
    - Dom(Telefone) = Telefone
- **ContratoCoquetel**
    - Dom(Coquetel) = Identificador de Festa
    - Dom(Funcionário) = CPF
    - Dom(HorasTrabalhadas) = Quantidade de Horas Trabalhadas por Funcionário
    - Dom(ValorPago) = Valor Pago a Funcionário
- **Funcionario**
    - Dom(CPF) = CPF
    - Dom(Nome) = Nome de Pessoa
    - Dom(RG) = RG
    - Dom(Estado) = Estado
    - Dom(Cidade) = Cidade
    - Dom(Rua) = Rua
    - Dom(Número) = Numeração de Rua
    - Dom(E-mail) = E-mail
    - Dom(TelefoneResidencial) = Telefone
    - Dom(TelefoneCelular) = Telefone
    - Dom(ValorPorHora) = Preço Pago a Funcionário por Hora
    - Dom(Função) = Função de Funcionário
- **ContratoFestFood**
    - Dom(FestFood) = Identificador de Festa
    - Dom(Segurança) = CPF
    - Dom(HorasTrabalhadas) = Quantidade de Horas Trabalhadas por Funcionário
    - Dom(ValorPago) = Valor Pago a Funcionário

Considerações do Mapeamento entre MER e Modelo Relacional

1. **Modelagem de Especificação: entidade Empresa**
    a) Optou-se por modelar tanto a entidade genérica Empresa quanto as entidades específicas: Fornecedora e Contratante;
    b) As entidades específicas foram modeladas devido ao fato de que elas são as únicas que participam de relacionamentos; além disso, para manter a consistência dos dados, devido à sobreposição e à quantidade de atributos que seriam replicados, a entidade genérica também foi modelada;
    c) Caso a entidade genérica fosse a única modelada, poderia haver um gasto de armazenamento aproximadamente igual (no uso de atributos booleanos) ou maior (no uso de uma nova tabela para armazenar os tipos) e, como apenas as empresas específicas participam dos relacionamentos, haveria um grande custo, em aplicação, para checar se a empresa que está participando do relacionamento é do tipo correto.

2. **Restrição de Especialização Total: entidade Empresa**
    a) Ela será garantida em aplicação.
3. **Modelagem de Especificação: entidade Funcionário**
    a) Optou-se por modelar apenas a entidade genérica Funcionário;
    b) A única especialização explícita de Funcionário é Segurança, que não apresenta nenhum atributo específico e só foi modelada separadamente no MER por ser a única função de Funcionário que trabalha nas Fest Food. Logo, o custo de consulta e junção de manter uma tabela separada de Segurança foi considerado maior do que fazer uma verificação para checar se todos os Funcionários que trabalham em uma Fest Food são Seguranças.
4. **Restrição de Especialização Total: entidade Festa**
    a) O atributo 'TipoFesta' é *not null* para garantir que toda festa tenha um tipo, porém, ainda assim, a garantia de que há uma tupla nas tabelas FestFood ou Coquetel para cada uma das tuplas presentes na tabela Festa será garantida em aplicação.
5. **Identificador Genérico: tabela Festa**
    a) Optou-se por incluir um identificador genérico para a tabela Festa, devido ao fato da quantidade de relacionamentos que ela participa com outras entidades, pois estes carregam as chaves primárias de Festa. Neste caso, o custo de memória para conter um atributo adicional compensa, devido ao ganho em consulta pela organização/busca do índice.
6. **Restrição de Participação Total: relacionamento Ocorre entre Fest Food e Local**
    a) A restrição de que toda Fest Food deve ocorrer em um Local, ou seja, que exista uma Locação relacionada a toda Fest Food, será garantida em aplicação.
7. **Identificador Genérico: tabela Locação**
    a) Optou-se por incluir um identificador genérico para a tabela Locação, devido ao fato dessa entidade dar origem à entidade fraca Lote, que, por isso, carregaria os três atributos que identificam cada Locação. O custo de memória para conter esse atributo adicional compensa devido ao ganho em consulta pela organização/busca do índice.
8. **Restrição de Participação Total: relacionamento Possui entre Fornecedora e CategoriaFornecimento**
    a) Ela será garantida em aplicação.
9. **Tabela ContratoFestFood: relacionamento Trabalha entre Fest Food e Segurança**
    a) A garantia de que um Funcionário que trabalha em uma Fest Food é especificamente um Segurança será feita em aplicação.
10. **Modelagem de Atributo Multivalorado: ProdutosFornecidos, no relacionamento Fornece entre Fornecedora e Coquetel**
    a) Optou-se por modelar o atributo como uma tabela a parte;
    b) Não existe uma definição específica da variedade e da quantidade de produtos fornecidos pela Empresa Fornecedora.
11. **Escolha de Chave: tabela ProdutosFornecidos**

a) O atributo Quantidade não foi incluído como parte da chave da tabela por questões semânticas, pois apenas o atributo Nome, juntamente com as chaves de Fornecedora e Coquetel, são suficientes para representar unicamente cada produto.

12. **Modelagem de Atributo Multivalorado: Telefones, da entidade Funcionario**
    a) Optou-se por modelar o atributo presente no MER como dois atributos monovalorados no modelo relacional: TelefoneCelular e TelefoneResidencial;
    b) Para tal considerou-se que, em um cenário real, uma pessoa fornece, usualmente, dois telefones no máximo; logo, optou-se por restringir os campos a apenas dois a fim de diminuir o gasto de disco e acelerar as consultas para os telefones dos funcionários.
13. **Modelagem de Atributo Multivalorado: Contatos, da entidade Empresa**
    a) Optou-se por modelar o atributo como uma tabela a parte, ContatoEmpresa;
    b) Não existe uma previsão do número de contatos que cada empresa possuirá.
14. **Escolha de Chave: tabela ContatoEmpresa**
    a) Foi escolhido o atributo E-mail para fazer parte da chave primária pois, para uma mesma empresa, dois contatos não terão o mesmo e-mail e, juntamente com a chave de Empresa, são suficientes para representar unicamente cada contato.
15. **Modelagem de Atributo Composto: Endereco, da entidade Local**
    a) Optou-se por modelar o atributo separadamente na tabela Local — ou seja, criando os atributos Cidade, Estado, Rua e Número — pois o atributo Cidade faz parte da chave primária de Local e necessita de consistência.
16. **Modelagem de Atributo Composto: Endereco, da entidade Funcionário**
    a) Optou-se por modelar o atributo separadamente na tabela Local — ou seja, criando os atributos Cidade, Estado, Rua e Número — pois uma das consultas principais do sistema (“dada uma cidade, informar os funcionários disponíveis nessa localidade”) é otimizada tendo o atributo Cidade sendo armazenado de forma separada;
    b) Caso todos os dados fossem condensados em um único atributo monovalorado, seria necessário um gasto grande de processamento de strings para realizar a consulta acima apresentada.
17. **Modelagem de Atributo Composto: Endereco, da entidade Empresa**
    a) Optou-se por modelar o atributo composto, no MER, como um único atributo monovalorado no modelo relacional, pois nem o contexto do sistema, nem as consultas importantes apresentam uma situação em que os dados precisariam estar separados (como rápido acesso em consultas), mostrando-se apenas como mais uma informação que será trazida a partir de consultas em outros atributos (como o CNPJ da empresa, por exemplo).
18. **Atributo Derivado: Orçamento, na tabela Coquetel**
    a) O valor deste atributo é calculado pela soma do preço de cada fornecimento de alimentos e bebidas (tabela FornecimentoCoquetel), pela diária da

locação do local (DiariaLocacao, na tabela Local) e pelo valor a ser pago para todos os funcionários contratados (tabela ContratoCoquetel).

19. **Restrição Semântica: relação entre Local e os tipos específicos de Festa**
    a) A garantia de que um mesmo Local não será alocado para duas festas distintas, na mesma data, será garantida em aplicação.
20. **Restrição Semântica: relação entre Segurança e os tipos específicos de Festa**
    a) A garantia de que um Segurança não poderá trabalhar em duas festas distintas, na mesma data, será garantida em aplicação.
21. **Atributo Derivado: ValorPago, nas tabelas ContratoCoquetel e ContratoFestFood**
    a) O valor deste atributo é calculado pelo produto dos atributos HorasTrabalhadas (presente em cada tupla das tabelas ContratoCoquetel e ContratoFestFood) e ValorPorHora (presente na tabela Funcionario).
22. **Atributo Composto: Dimensoes, da entidade Lote**
    a) Optou-se por modelar o atributo separadamente na tabela Lote — ou seja, criando os atributos Largura e Comprimento — para permitir que consultas sobre os tamanhos unidimensionais dos lotes sejam possíveis e otimizadas.
23. **Restrição Semântica: atributo Função da tabela Funcionário**
    a) O atributo Função foi colocado como *not null* para garantir que todo funcionário tenha uma função específica, porém, Segurança é a única especialização explícita no MER.

IMPLEMENTAÇÃO DO PROJETO

Para a implementação do projeto apresentado, foi escolhido utilizar o sistema gerenciador de banco de dados (SGBD) Oracle, por meio da sua instância na nuvem do ICMC. Além disso, um software para manipulação dos dados foi desenvolvido na linguagem Java, apoiado pela IDE Eclipse e com a interface gráfica feita por meio da biblioteca JavaFX.

Como requisitos de sistema necessários para executar o software corretamente, é preciso um sistema operacional com o Java Runtime Environment (JRE) devidamente instalado — disponível para Windows, Mac OS e distribuições Linux — e conexão estável com a internet, para permitir o acesso ao banco de dados na nuvem. A interface desenvolvida apresenta uma tela de login, que permite conectar à instância do Oracle disponível na nuvem do ICMC por meio de um usuário e senha já existentes no sistema.

Gatilhos, procedimentos e funções foram implementados, em SQL, para: (a) permitir que o campo ID das tabelas FESTA e LOCACAO sejam auto-incrementados, em conjunto com estruturas SEQUENCE; (b) calcular o atributo VALORPAGO das tabelas CONTRATOCOQUETEL e CONTRATOFESTFOOD, por ser atributo derivado no MER; (c) calcular o atributo ORCAMENTO da tabela COQUETEL, por ser atributo derivado no MER.

Os arquivos anexos a este relatório apresentam scripts na linguagem SQL para a criação de todas as tabelas do banco de dados (FF_CreateTables.sql), para compilação de gatilhos e procedimentos em PL/SQL (FF_Triggers.sql) e para a inserção de tuplas iniciais (FF_InitialData.sql). Os demais comandos SQL utilizados no projeto, ou seja, os que são utilizados dentro da interface para inserção, remoção, alteração e busca de tuplas serão apresentadas nas próximas subseções.

Para viabilizar o processamento dos dados e a visualização deles pelo JavaFX, foi criada uma classe para cada tabela presente no banco de dados. Todas elas fazem parte do pacote `backend.tables` e apresentam todos os métodos utilizados para manipular os dados.

Visualização de tabela inteira

Todas as tabelas podem ser visualizadas por completo por meio da funcionalidade "Visualizar Registros" presente na interface gráfica do software. A busca por esses dados é feito por meia da função estática `tableView()`, presente em todas as classes relacionadas à tabelas, a partir de um comando SQL simples utilizando o comando `SELECT *`. Abaixo, é possível ver um exemplo desta função, com o comando SQL em destaque, relacionada à tabela AtribuicaoCategoria.

src/backend/AtribuicaoCategoria.java

```
public static ObservableList<AtribuicaoCategoria> tableView(){
          ResultSet res;
          List<AtribuicaoCategoria> list = new
ArrayList<AtribuicaoCategoria>();
          String sql ="select * from ATRIBUICAOCATEGORIA";
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new AtribuicaoCategoria(
res.getString(1), res.getString(2), res.getString(3)));

               res.close();
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          } catch (SQLException e) {
            throw new RuntimeException(e);
          }
     }
```

Inserção de novas tuplas

Todas as tabelas podem ter novas tuplas inseridas por meio da funcionalidade “Inserir Registros” presente na interface gráfica do software. O processo de inserção é feito por meio da função estática `insert<nome da tabela>()`, presente em todas as classes relacionadas à tabelas, a partir de um comando SQL simples utilizando o comando INSERT. Parte do comando é escrito por meio da função `toString()` do objeto instanciado a partir da tabela. Erros que possam acontecer durante a inserção são tratados e levados ao usuário pela interface. Abaixo, é possível ver um exemplo desta função, com o comando SQL em destaque, relacionada à tabela Convidado.

src/backend/Convidado.java

```
public static void insertConvidado(Convidado convidado) throws
Exception {
          String sql = "insert into CONVIDADO (EMAIL, NOME,
TELEFONE) values("+convidado+")";
          try {
               ConnectionManager.query(sql);
               ConnectionManager.closeQuery();
          }catch(SQLException e){
               String mesg="";
               String aux = e.getMessage().split("[:(). ]")[0];
               if(aux.equals("ORA-00001")){
                         mesg = "Já há um Convidado com esse
Email.Por favor digite outro Email.";
               }else if(aux.equals("ORA-01400")) {
                         mesg = "Os campos Email e Nome tem que
ser preenchidos.";
               }else if(aux.equals("ORA-12899")) {
                         mesg = "Os limites de caracteres dos
campos são: Nome - 60; Email - 60; Telefone - 11.";
               }
               throw new Exception(mesg);
          }catch(Exception e) {
               throw new RuntimeException();
          }
     }

     @Override
     public String toString() {
          return
"'"+this.email+"','"+this.nome+"','"+this.telefone+"'";
     }
```

Remoção de tuplas

Todas as tabelas podem ter tuplas removidas por meio da funcionalidade “Remover Registros” presente na interface gráfica do software. A remoção é feita por meio da função estática `delete<nome da tabela>()`, presente em todas as classes relacionadas à tabelas, a partir de um comando SQL simples utilizando o comando DELETE. Parte do comando é escrito por meio da função `toStringRestritions()` do objeto instanciado a partir da tabela. Erros que possam acontecer durante a inserção são tratados e levados ao usuário pela interface. Abaixo, é possível ver um exemplo desta função, com o comando SQL em destaque, relacionada à tabela FornecimentoCoquetel.

src/backend/FornecimentoCoquetel.java

```
public static void
deleteFornecimentoCoquetel(FornecimentoCoquetel
fornecimentoCoquetel) throws Exception {
          String aux =
fornecimentoCoquetel.toStringRestritions();
          if(aux.equals(" ")) {
               throw new Exception("É necessário preencher pelo
menos 1 dos campos identificadores do registro a remover.");
          }
          String sql = "delete from FORNECIMENTOCOQUETEL"+aux;
          try {
               ConnectionManager.query(sql);
               ConnectionManager.closeQuery();
          }catch(SQLException e) {
               throw new RuntimeException();
          }
     }

private String toStringRestritions() {
          String res = " where ";
          if(!fornecedora.equals("")) {
               res += " FORNECEDORA = '"+this.fornecedora+"'";
          }
          if(coquetel != 0) {
               if(!res.equals(" where "))
                    res += " and ";
               res += " COQUETEL = "+this.coquetel;
          }
          if(res.equals(" where "))
               res = " ";
          return res;
     }
```

Atualização de tuplas

Todas as tabelas podem ter tuplas atualizadas por meio da funcionalidade “Atualizar Registros” presente na interface gráfica do software. A remoção é feita por meio da função estática update<nome da tabela>(), presente em todas as classes relacionadas à tabelas, a partir de um comando SQL simples utilizando o comando UPDATE. Parte do comando é escrito por meio da função toStringUpdates() do objeto instanciado a partir da tabela. Erros que possam acontecer durante a inserção são tratados e levados ao usuário pela interface. Abaixo, é possível ver um exemplo desta função, com o comando SQL em destaque, relacionada à tabela ContratoFestFood.

src/backend/ContratoFestFood.java

```
public static void updateContratoFestFood(ContratoFestFood
contratoFestFood) throws Exception {
          String sql = "update CONTRATOFESTFOOD set"
                    + contratoFestFood.toStringUpdates()
                    + " where FESTFOOD =
"+contratoFestFood.festFood
                    + " and SEGURANCA =
'"+contratoFestFood.seguranca+"'";
          try {
               ConnectionManager.query(sql);
               ConnectionManager.closeQuery();
          }catch(SQLException e){
               String mesg="";
               String aux = e.getMessage().split("[:(). ]")[0];
               if(aux.equals("ORA-01747")){
                    mesg = "É necessário preencher pelo menos 1
dos campos a alterar.";
               }
               throw new Exception(mesg);
          }catch(Exception e) {
               throw new RuntimeException();
          }
     }

private String toStringUpdates() {
          String res = "";
          if(horasTrabalhadas != 0) {
               res += " HORASTRABALHADAS =
"+this.horasTrabalhadas;
          }
          return res;
     }
```

Consultas avançadas

A interface de usuário está preparada para realizar seis consultas de complexidade média/alta, trazendo informações que possam ser úteis e que dependam de mais de uma tabela. Abaixo, são apresentadas essas consultas e os comandos SQL que as implementam (eles também estão disponíveis no arquivo FF_AdvancedQueries.sql, junto com os demais scripts SQL).

1) **LOTES DA LOCAÇÃO**
   Indicada uma locação, informe todos os lotes em que ela está dividida e, caso seja possível, a empresa fornecedora que a adquiriu.

src/advancedQueries/LotesLocacao.java

```
public static ObservableList<LotesLocacao> tableView(int
locacao){
          ResultSet res;
          List<LotesLocacao> list = new
ArrayList<LotesLocacao>();
          String sql = "select LT.NUMERO, LT.PRECO, LT.LARGURA,
LT.COMPRIMENTO, EM.NOMEFANTASIA, LT.FORNECEDORA " +
          "from LOTE LT " +
          "left join EMPRESA EM on LT.FORNECEDORA = EM.CNPJ " +
          "where LOCACAO = "+locacao;
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new LotesLocacao(res.getInt(1),
res.getFloat(2), res.getFloat(3), res.getFloat(4),
res.getString(5), res.getString(6)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e);
          }
     }
```

2) **CUSTO DOS COQUETÉIS**
   Mostrar o custo total de cada coquetel, apenas para os que têm um número mínimo de convidados.

src/advancedQueries/CustoCoquetel.java

```
public static ObservableList<CustoCoquetel> tableView(int
minConv){
```

```
          ResultSet res;
          List<CustoCoquetel> list = new
ArrayList<CustoCoquetel>();
          String sql ="select F.NOME, F.CONTRATANTE, F.DATA,
count(CON.CONVIDADO) as NOCONVIDADOS, COQ.ORCAMENTO " +
"from COQUETEL COQ " +
"inner join CONVITE CON on COQ.FESTA = CON.COQUETEL " +
"inner join FESTA F on F.ID = COQ.FESTA " +
"group by F.ID, F.NOME, F.CONTRATANTE, F.DATA, COQ.ORCAMENTO " +
"having count(CON.CONVIDADO) >= "+minConv;
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new CustoCoquetel(res.getString(1),
res.getString(2), res.getString(3), res.getInt(4),
res.getFloat(5)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e.getMessage());
          }
     }
```

3) **FOLHA DE PAGAMENTO DO FUNCIONÁRIO**
   Indicado um funcionário e um intervalo de datas, informar em quais festas trabalhou e quanto deve receber por cada uma.

src/advancedQueries/PagamentoFuncionario.java

```
public static ObservableList<PagamentoFuncionario>
tableView(String cpfFuncionario, String dataInic, String
dataFinal){
          ResultSet res;
          List<PagamentoFuncionario> list = new
ArrayList<PagamentoFuncionario>();
          String sql = "select F.NOME, F.DATA, F.CONTRATANTE,
F.TIPOFESTA, C.VALORPAGO " +
"from FESTA F, CONTRATOCOQUETEL C " +
"where F.ID = C.COQUETEL " +
"and C.FUNCIONARIO = '"+cpfFuncionario+"' "+
"and F.DATA >= to_date('"+dataInic+"', 'mm/yyyy') " +
"and F.DATA <= to_date('"+dataFinal+"', 'mm/yyyy') " +
"union "+
"select F.NOME, F.DATA, F.CONTRATANTE, F.TIPOFESTA, C.VALORPAGO "
+
"from FESTA F, CONTRATOFESTFOOD C " +
"where F.ID = C.FESTFOOD " +
"and C.SEGURANCA = '"+cpfFuncionario+"' " +
```

```
"and F.DATA >= to_date('"+dataInic+"', 'mm/yyyy') " +
"and F.DATA <= to_date('"+dataFinal+"', 'mm/yyyy')";
          try{
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new
PagamentoFuncionario(res.getString(1),res.getString(2),res.getStr
ing(3),res.getString(4),res.getFloat(5)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e);
          }
     }
```

4) MÉDIA SALARIAL POR ESTADO

Indicado um estado do país, informar o número de funcionários disponíveis e o custo médio por hora para cada função que um empregado pode exercer.

src/advancedQueries/SalarioPorEstado.java

```
public static ObservableList<SalarioPorEstado> tableView(String
estado){
          ResultSet res;
          List<SalarioPorEstado> list = new
ArrayList<SalarioPorEstado>();
          String sql = "select FUNCAO, count(*) as QUANTIDADE,
avg(VALORPORHORA) as MEDIASALARIAL " +
               "from FUNCIONARIO " +
               "where ESTADO = '"+estado+"' " +
               "group by FUNCAO";
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new
SalarioPorEstado(res.getString(1), res.getInt(2),
res.getFloat(3)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e);
          }
     }
```

5) **FAIXA DE PREÇO FREQUENTE**

Indicado uma categoria de fornecimento de produtos, informar as empresas fornecedoras que trabalham com essa categoria na faixa de preço mais comum para aquela categoria.

src/advancedQueries/FaixaPrecoFrequente.java

```
public static ObservableList<FaixaPrecoFrequente>
tableView(String categoria){
          ResultSet res;
          List<FaixaPrecoFrequente> list = new
ArrayList<FaixaPrecoFrequente>();
          String sql = "select E.NOMEFANTASIA, F.CNPJ,
ATR.FAIXAPRECO " +
     "from FORNECEDORA F, ATRIBUICAOCATEGORIA ATR, EMPRESA E " +
     "where E.CNPJ = F.CNPJ " +
     "and F.CNPJ = ATR.FORNECEDORA " +
     "and ATR.CATEGORIA = '"+categoria+"' " +
     "and ATR.FAIXAPRECO = " +
     "(select STATS_MODE(FAIXAPRECO) from ATRIBUICAOCATEGORIA
where CATEGORIA = '"+categoria+"')";
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new
FaixaPrecoFrequente(res.getString(1), res.getString(2),
res.getString(3)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e);
          }
     }
```

6) **PRÓXIMAS FEST FOODS**

Informar todas as próximas Fest Foods (ou seja, as que ainda não aconteceram), junto com o número de lotes disponíveis para ela, o número de lotes já adquiridos e o número de ingressos vendidos

src/advancedQueries/ProximasFestFoods.java

```
public static ObservableList<ProximasFestFoods> tableView(){
          ResultSet res;
          List<ProximasFestFoods> list = new
ArrayList<ProximasFestFoods>();
          String sql = "with " +
"QING as " +
"(select FESTFOOD, count(*) as INGRESSOSVENDIDOS " +
```

```
"from INGRESSO " +
"group by FESTFOOD), " +
"QLOTE as " +
"(select LOCA.FESTFOOD, count(LOTE.NUMERO) as TOTALLOTES,
count(LOTE.FORNECEDORA) as LOTESVENDIDOS " +
"from LOCACAO LOCA, LOTE " +
"where LOCA.ID = LOTE.LOCACAO " +
"group by LOCA.FESTFOOD) " +
"select F.NOME, F.DATA, F.CONTRATANTE, QLOTE.TOTALLOTES,
QLOTE.LOTESVENDIDOS, QING.INGRESSOSVENDIDOS " +
"from FESTA F, FESTFOOD FF, QING, QLOTE " +
"where F.ID = FF.FESTA " +
"and FF.FESTA = QING.FESTFOOD " +
"and QING.FESTFOOD = QLOTE.FESTFOOD "+
"and F.DATA > SYSDATE";
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(new
ProximasFestFoods(res.getString(1), res.getString(2),
res.getString(3), res.getInt(4), res.getInt(5), res.getInt(6)));
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e) {
               throw new RuntimeException(e);
          }
     }
```

Consultas extras para outras operações

Para garantir consistência em certas operações, outras consultas SQL são realizadas além destas já mostradas para abastecer campos do tipo ComboBox na interface — ou seja, ao invés do usuário digitar um valor que pode não ser válido, ele apenas escolhe o que ele deseja em uma lista de valores válidos. Abaixo, como exemplo, é mostrada a função `getListaSeguranca()`, na classe Funcionario, que recupera os dados dos seguranças cadastrados no banco de dados para que sejam feitos contratos de trabalho em Fest Foods apenas com empregados com esta função.

src/backend/Funcionario.java

```
public static ObservableList<String> getListaSeguranca(){
          ResultSet res;
          List<String> list = new ArrayList<String>();
          String sql="select F.NOME,F.CPF from FUNCIONARIO F
where (upper(FUNCAO) = 'SEGURANÇA' OR upper(FUNCAO) =
'SEGURANCA')";
```

```
          try {
               res = ConnectionManager.query(sql);
               while(res.next())
                    list.add(res.getString(1)+" /
"+res.getString(2));
               res.close();
               ConnectionManager.closeQuery();

               return FXCollections.observableList(list);
          }catch(SQLException e){
               throw new RuntimeException(e);
          }
     }
```

Não é viável apresentar todos os códigos em cópia neste relatório, mas a lista abaixo indica todas as funções criadas para esse fim e o arquivo de código em que se localizam no código-fonte:

- **FestFood.java** — getListaFestFood()
- **Coquetel.java** — getListaCoquetel()
- **Fornecedora.java** — getListaFornecedora()
- **Empresa.java** — getListaEmpresa()
- **CategoriaFornecimento.java** — getListaCategoria()
- **Contratante.java** — getListaContratante()
- **Local.java** — getListaLocal(), getListaNome(), getListaId()
- **Convidado.java** — getListaConvidado()
- **Locacao.java** — getListaLocacao(), getListaId()
- **Funcionario.java** — getListaFuncionario(), getListaSeguranca(), getListaCpf(), getListaEstado()
- **ContatoEmpresa.java** — getListaEmail()

Todos os arquivos do projeto, incluindo código-fonte, scripts SQL e esse relatório, estarão disponíveis no Escaninho do aluno David Souza Rodrigues (nº USP 4461180).

CONCLUSÃO

O projeto mostrou-se de uma grande importância para o aprendizado dos conteúdos apresentados durante o curso de Bases de Dados. Em cada parte do projeto, foi possível aplicar as técnicas e a teoria vistas em aula. A maior dificuldade residiu na implementação da aplicação, devida à complexidade do software, que demanda mais tempo de trabalho e apresenta mais detalhes a serem programados e conferidos.

Um problema que atrasou o desenvolvimento do projeto, ainda no início e que pode ser corrigido para as próximas turmas, é a comunicação na definição da temática. Por ser um trabalho sobre “Assessoria de Eventos”, era muito claro que tratavam-se de eventos no geral e, apenas posteriormente, foi falado que teriam de ser eventos que parecessem

mais com “festas”, fazendo o grupo perder uma semana e meio de trabalho. Imagino que isso sendo melhor definido, sem ambiguidades, possa ajudar o trabalho nas próximas turmas.