Banco de Dados

Jones Cavalcanti Sarmento

Curso Técnico em Desenvolvimento de Sistemas
Educação a Distância
2019

Banco de Dados

Jones Cavalcanti Sarmento

Curso Técnico em Informática

Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa

Educação a Distância

Recife - PE

2019

Licença Pública Creative Commons
Atribuição-NãoComercial-Compartilhalgual 4.0 Internacional

Professor(es) Autor(es)
Jones Cavalcanti Sarmento

Revisão
Jones Cavalcanti Sarmento
José Américo

Coordenação de Curso
José Américo

Coordenação Design Educacional
Deisiane Gomes Bazante

Design Educacional
Ana Cristina do Amaral e Silva Jaeger
Fernanda Paiva Furtado da Silveira
Izabela Pereira Cavalcanti
Jailson Miranda
Roberto de Freitas Morais Sobrinho

Descrição de imagens
Sunnye Rose Carlos Gomes

Catalogação e Normalização
Hugo Cavalcanti (Crb-4 2129)

Diagramação
Jailson Miranda

Coordenação Executiva
George Bento Catunda
Renata Marques de Otero
Manoel Vanderley dos Santos Neto

Coordenação Geral
Maria de Araújo Medeiros Souza
Maria de Lourdes Cordeiro Marques

Secretaria Executiva de
Educação Integral e Profissional

Escola Técnica Estadual
Professor Antônio Carlos Gomes da Costa

Gerência de Educação a distância

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISDB

S246b

Sarmento, Jones Cavalcanti.
Banco de Dados: Curso Técnico em Desenvolvimento de Sistemas: Educação a distância / Jones Cavalcanti Sarmento. – Recife: Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa, 2019.
77 p.: il.

Inclui referências bibliográficas.
Caderno eletrônico produzido em setembro de 2019 pela Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa.

1. Banco de dados — Gerenciamento — Programas de computador. 2. Projeto de banco de dados. 3. Software de aplicativos — Desenvolvimento. I. Sarmento, Jones Cavalcanti. II. Título.

CDU – 004.658

Elaborado por Hugo Carlos Cavalcanti | CRB-4 2129

Sumário

Introdução 6

1.Competência 01 | Conhecer os Princípios de Banco de Dados 8

1.1 Fundamentos de Bancos de Dados Relacionais 8

1.2 Um Banco de Dados de Exemplo 13

1.3 Structured Query Language (SQL) 17

2.Competência 02 | Elaborar um Modelo de Entidade-Relacionamento 27

2.1 Modelagem de Banco de Dados 27

2.2 Modelo ER e ER Estendido 30

2.3 UML 37

2.4 Ferramenta CASE – EERCASE 39

2.5 Ferramenta CASE – StarUML 41

2.6 Transformação de Modelo Conceitual para SQL 43

2.7 Dependência Funcional 50

2.8 Formas Normais 52

2.8.1 Primeira Forma Normal 52

2.8.2 Segunda Forma Normal 53

2.8.3 Terceira Forma Normal 55

3.Competência 03 | Construir Tabelas e Dicionário de Dados de um Banco de Dados 57

3.1 Instalação do Ambiente e Criação do Banco de Dados Vendas 57

3.1.1 Instalação do SQLite 57

3.1.2 Instalação do PostgreSQL 59

3.2 Scripts do Banco de dados Vendas 60

3.2.1 Scripts SQL-DDL-CREATE, BD Vendas 60

3.2.2 Scripts SQL-DML-INSERT, BD Vendas 62

3.2.3 Scripts SQL-DML-SELECT, BD Vendas 67

3.2.4 Scripts SQL-DML-UPDATE, BD Vendas 71

3.2.5 Scripts SQL-DML-DELETE, BD Vendas 73

3.2.6 Scripts SQL-DDL-DROP, BD Vendas 73

Conclusão 74

Referências 75

Minicurrículo do Professor 76

Introdução

A disciplina de banco de dados tem o objetivo de levar para você os principais conceitos relacionados a projeto, desenvolvimento e manutenção de bancos de dados. Esta disciplina vai proporcionar, em primeiro momento, acesso aos dados, que são os responsáveis por movimentações bancárias, vendas de produtos, sistemas de mensagens, entre outras muitas aplicações. Esperamos que você se identifique com o conteúdo e desenvolva com sucesso esta disciplina. Bons estudos!

Bancos de Dados (BD) e Sistemas Gerenciadores de Bancos de Dados (SGBD) estão presentes na maioria dos sistemas computacionais, uma vez que é necessário consultar e armazenar, de forma eficiente, transações feitas por usuários diversos. Um exemplo claro da necessidade de um BD confiável é quando um cliente faz uma compra em um site de e-commerce por meio de cartão de crédito. Neste caso, os produtos são cadastrados pela empresa em um sistema de banco de dados de forma que deve reproduzir a real situação do seu estoque. Assim também, o usuário deve efetuar um cadastro no site inserindo seus dados pessoais, e-mail e intenções de compra; que, posteriormente, serão utilizados pela empresa como um meio de mala direta, propondo futuras promoções em estratégias de marketing. Os dados do cartão de crédito devem ser consultados em um banco de dados, para que sua compra seja aprovada pela operadora. A própria compra deve ser armazenada, tanto em banco de dados da empresa para fins de controle e processos corporativos, como também na Fazenda para fins de tributação. Imagine se não houvesse uma garantia na transação, como seria?

Perceba que esse cenário, comum nos dias de hoje, envolve vários bancos de dados. Esses tipos de BD são conhecidos como bancos de dados tradicionais e transacionais, pois produzem dados simples (textos e números) que são decorrentes de transações. No entanto, existem outros tipos de bancos de dados, como por exemplo os BD de multimídia (vídeos, imagens e sons); data warehouse (multidimensionais - para apoio à decisão empresarial) e BD geográficos (imagens de satélite) (SARMENTO, 2015).

Embora existam estes outros tipos de bancos de dados, ou seja, BD não convencionais, você deve absorver os conceitos tradicionais que serão abordados nesta disciplina. Ao final o você estará apto a desenvolver sistemas de computação com acesso a bancos de dados, desenvolver consultas por meio de instruções SQL (Structured Query Language) e elaborar modelos de dados eficientes. Neste sentido, na competência 01, abordaremos os fundamentos, bancos de dados relacionais e SQL. Na competência 02, utilizando ferramentas do tipo CASE (Engenharia de Software

Assistida por Computador), você vai aprender a modelar um banco de dados passando por diferentes níveis de abstração, utilizando linguagens diagramáticas como o modelo EER (Entidade-Relacionamento Estendido) e a linguagem UML (Linguagem de Modelagem Unificada). Na competência 3 você vai instalar, criar um banco de dados de exemplo e scripts em SQL, que serão desenvolvidos a partir dos artefatos produzidos na competência 2.

Uma dica para você: dedique-se ao máximo no aprendizado desta disciplina, pois se trata de uma matéria indispensável para sistemas de computação. Outros conceitos estão relacionados aos que veremos aqui, como por exemplo, Big Data, Sistemas para Apoio à Decisão e Descoberta do Conhecimento, que estão em evidência nos dias atuais e necessitam de profissionais especializados.

Você está pronto para um mundo de dados? Então venha conosco!

1.Competência 01 | Conhecer os Princípios de Banco de Dados

Nesta primeira competência veremos os principais conceitos de bancos de dados relacionais. Embora existam conceitos genéricos acerca de bancos de dados (BD), utilizaremos os conceitos que já são consolidados pela academia e pelas empresas, que são os BD que contêm relacionamentos entre tabelas, chamados Bancos de Dados Relacionais. Os sistemas de computação acessam esses bancos de dados com o objetivo de construir algum negócio e/ou tomar alguma decisão corporativa, com base nos fatos ocorridos em transações. Nos materiais complementares você vai encontrar outras fontes que ajudarão a construir uma análise mais completa acerca de outros tipos de BD, como exemplo Bancos de Dados Orientado a Objetos.

1.1 Fundamentos de Bancos de Dados Relacionais

Antes de iniciarmos efetivamente a disciplina de banco de dados, é necessário que você aprenda alguns conceitos básicos que nortearão o restante deste e-book. Por exemplo, observe esta frase e tente compreender o que se diz: “Sistemas Gerenciadores de Bancos de Dados armazenam, gerenciam transações e disponibilizam Dados de diferentes tipos em diferentes Bancos de Dados”. Parece uma frase redundante e complexa, não? Contudo, na frase destacada temos três principais conceitos que você precisa entender e distingui-los: **Sistemas Gerenciadores de Bancos de Dados (SGBD)**, **Bancos de Dados (BD)** e, por último, não menos importante e a razão de existir este *e-book*, os **Dados**. Segundo os autores Elmasri e Navathe (2005), um Banco de Dados é uma coleção de dados relacionados. Dados, por sua vez, são fatos que podem ser gravados e que possuem algum significado para o negócio de quem os gravou. Já os SGBD´s são sistemas de computador usados para definir, gerenciar e garantir o armazenamento de bancos de dados que contêm dados de diversos tipos e origens. Em termos práticos, por exemplo, uma instituição como o IBGE (Instituto Brasileiro de Geografia e Estatística) que coleta informações das pessoas para recenseamento, os dados são um conjunto de caracteres inseridos em campos de um formulário: Nome, Telefone, Endereço e Data Nascimento. O banco de dados será formado por todos os formulários agrupados e relacionados, que neste caso, tem-se um banco de dados de recenseamento. O SGBD será o sistema (e.g. MySQL®, Oracle®, PostgreSQL® e SQL Server®) que vai armazenar esse banco de dados e garantir que os dados estejam salvos e disponíveis. Embora sejam conceitos diferentes, muitas pessoas confundem o termo

banco de dados (relações de dados) com SGBD (sistemas de software). Adotaremos os SGBD´s **SQLite®** e **PostgreSQL® (versão 9.6.12)**, que serão abordados na terceira Competência.

Os Sistemas Gerenciadores de Bancos de Dados atuais utilizam largamente o modelo relacional de dados, que são dados armazenados em uma coleção de **Tabelas** contendo **Colunas** e **Linhas**. Cada coluna é um atributo desta entidade, e cada linha da tabela representa uma tupla (registro) de uma **Entidade** do mundo real. Entende-se como entidade qualquer objeto do mundo real que contenha algum atributo. Como exemplo, vamos utilizar uma Pessoa (um humano) como nossa primeira entidade, e para armazenar seus dados, utilizaremos a tabela chamada Pessoa (simples, não?). Seus atributos (que serão as colunas da tabela) serão: “CPF”, “Nome”, “Telefone” e “Data de Nascimento”, ou seja, características que uma pessoa têm. Ao definir os atributos, e inserir dados para uma pessoa, cada linha/tupla forma uma instância desta pessoa: CPF: “889.458.665-90”, Nome: “Fulano de Tal das Camborja”, Telefone: “81-9988877XX”, Data de Nascimento: “23 de maio de 1993”. A Figura 1 mostra o exemplo da tabela “Pessoa” citada.

Atributos

PESSOA

CPF	Nome	Telefone	Data_Nascimento
889.458.665-90	Fulano de Tal das Camborja	81-9988877xx	23/05/1993
005.884.632-52	Beltrano das Bruguelas Coimbra	21-9987455xx	15/07/1984
448.997.564-23	Cicrano Belmonte Xavier	11-9865542xx	20/08/1992
548.654.789-53	Ciriguela Paulista Cidreira	54-9547777xx	30/03/1987

Tuplas

Figura 1 – Tabela de Exemplo “Pessoa”.
Fonte: O Autor (2019).
Descrição: Exemplo de uma tabela em bancos de dados relacional com alguns registros, tais como: CPF, nome, número de telefone e data de nascimento.

Existem outros modelos de bancos de dados, como os Hierárquicos e os de Rede. Estes bancos de dados tiveram seus dias de glória no passado, no entanto, com o advento do modelo Relacional trazendo facilidades de uso, foram aos poucos sendo substituídos.

Para fins de fixação de conteúdo, você deve realizar um exercício em sua casa criando tabelas contendo tuplas e atributos. Por exemplo, uma tabela chamada “Carros” com os atributos:

“Cor”, “Ano”, “Marca” e “Modelo”. Ao final, insira alguns dados fictícios nestas tabelas. Este exercício faz com que você entenda a estruturação básica e a necessidade de organizar os dados da melhor forma possível.

Ao criar tabelas com atributos variados você vai notar que, em determinado momento, não faz sentido criar uma tabela enorme com uma infinidade de atributos contendo dados repetidos. A tabela “Pessoa” poderia conter atributos, por exemplo, referentes ao seu endereço. No entanto, em bancos de dados relacionais é comum haver inúmeras tabelas relacionadas umas com as outras. Essas relações proporcionam uma melhor organização dos dados, tanto no momento de inseri-los, quanto no momento de consultá-los. Esta organização evita, também, que um determinado dado se repita, conceitualmente isto é chamado de redundância de dados.

Para que haja um relacionamento entre duas tabelas, é preciso que ambas tenham um atributo em comum. Esse atributo é chamado de chave (ou superchave), que é um valor único, identificador do registro na tabela e é amplamente conhecido como **Chave Primária** ou PK (Primary Key). A chave primária serve, também, para evitar que registros iguais se repitam. Por exemplo, na tabela “Pessoa”, uma chave capaz de identificar um registro é o “CPF”. Neste caso, ao tentar inserir duas vezes a mesma pessoa com o mesmo CPF na tabela, o SGBD deve identificar a duplicidade de informações e evitar que a operação se concretize, uma vez que números de CPF são únicos para cada pessoa. Este conceito é conhecido como integridade de chave. A figura 2 mostra um exemplo de chave primária e uma relação entre as tabelas “Pessoa” e “Endereço”.

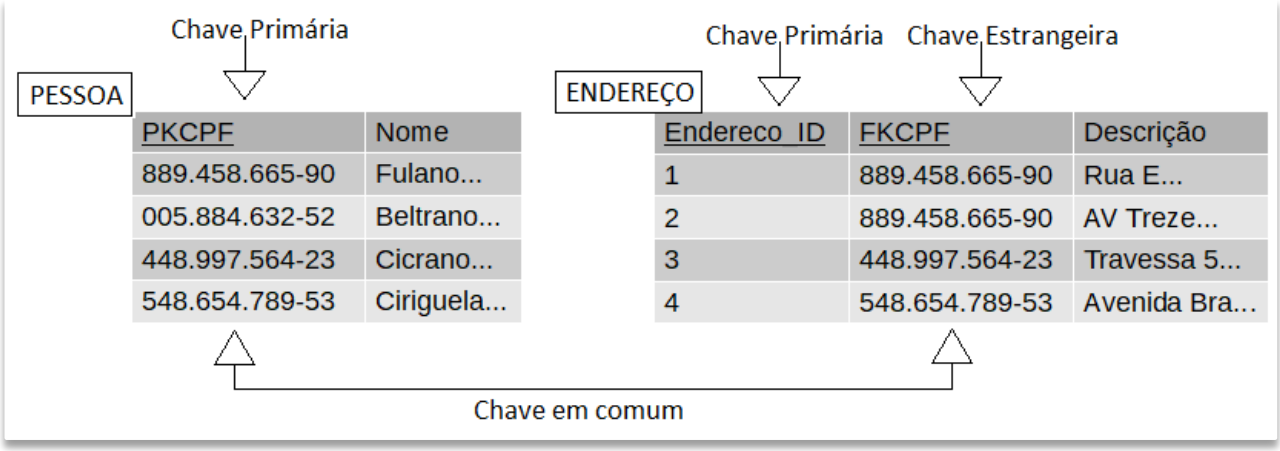

Figura 2 - Atributo em comum entre as tabelas “Pessoa” e “Endereço”.Fonte: O Autor (2019).
Descrição: A figura apresenta duas tabelas chamadas, “Pessoa” e “Endereço”. Ambas contêm um atributo em comum que é o CPF. Na tabela “Pessoa” o “CPF” aparece como chave primária. Na tabela “Endereço” o “CPF” aparece como chave estrangeira. As tabelas estão se relacionando por meio do CPF.

Na Figura 2 verifica-se também o conceito de **Chave Estrangeira** (Foreign Key – FK). Uma FK é um atributo que tem seu valor como chave primária em outra tabela, fazendo com isso uma ligação entre duas tabelas. No exemplo da Figura 2, observe que a pessoa identificada com o CPF "889.458.665-90" possui dois endereços inseridos na tabela "Endereço": "Endereco_ID" 1 e 2. Esta estrutura evita que a tabela "Pessoa" tenha registros repetidos (observe na Figura 3) ou então, evita-se que sejam adicionadas (na tabela "Pessoa") várias colunas informando o endereço: Endereço1, Endereço2..., EndereçoN (observe na Figura 4). E se uma pessoa tiver cinco endereços diferentes? Ou aquelas que têm 'N' endereços distintos? Caso sejam criadas várias colunas como citado, e a pessoa tenha apenas um endereço, as outras colunas da tabela ficarão vazias (conhecido como valor NULL), o que configura uma prática a ser evitada em banco de dados relacional por motivo de implementação e otimização de sistemas (observe a Figura 4).

PESSOA

PKCPF	Nome	Endereço
889.458.665-90	Fulano...	Rua E...
889.458.665-90	Fulano...	AV Treze...
005.884.632-52	Beltrano...	NULL
448.997.564-23	Cicrano...	Travessa 5...
548.654.789-53	Ciriguela...	Avenida Bra...

Figura 3 - Erro de redundância de dados.
Fonte: O Autor (2019).
Descrição: A figura é uma tabela contendo três atributos: Chave primária PKCPF, Nome e Endereço. A tabela contém dados fictícios com um exemplo de erro a ser evitado que é a redundância de dados, nesse caso a mesma pessoa com mais de um endereço.

PESSOA

PKCPF	Nome	Endereço1	Endereço2	EndereçoN
889.458.665-90	Fulano...	Rua E...	AV Treze...	NULL
005.884.632-52	Beltrano...	NULL	NULL	NULL
448.997.564-23	Cicrano...	Travessa 5...	NULL	NULL
548.654.789-53	Ciriguela...	Avenida Bra...	NULL	NULL

Figura 4 - Exemplo de erro em tabela de banco de dados relacional**Fonte:** O Autor (2019).
Descrição: A tabela contém cinco atributos: Chave primária PKCPF, Nome, Endereço1, Endereço2 e EndereçoN. A tabela contém dados fictícios com um exemplo de erro a ser evitado, que são dados nulos inseridos em colunas criadas em

tabela sem relacionamento.

Para evitar redundâncias, se faz necessária a criação de uma relação de 1 para N, ou seja, uma pessoa tem 1 ou vários endereço(s). A chave primária da tabela do lado '1' vai ser usada como chave estrangeira da tabela no lado 'N'. Na próxima competência abordaremos melhor os conceitos de relacionamentos entre tabelas.

O Sistema Gerenciador de Banco de Dados tem um papel fundamental para garantir que as relações estejam consistentes e íntegras. Ao fazer uma operação de inserção de dados em um banco de dados que contenha mais de uma tabela relacionada, o SGBD precisa garantir que o valor de chave primária de uma tabela corresponde ao valor de chave estrangeira de outra. Tomando como exemplo as tabelas "Pessoa" e "Endereço" vistas na Figura 2, ao inserir dados na tabela Endereço, o SGBD precisa verificar se existe um registro na tabela "Pessoa" para este endereço e, se não existir, esta operação não deve ser concluída, uma vez que não existe endereço sem um dono. Se assim for, teremos uma tabela com endereços sem seus respectivos moradores, o que pode causar uma verdadeira inconsistência nos dados. Este conceito é conhecido como Integridade Referencial, onde cada chave estrangeira deve ter seu respectivo referente como chave primária.

Observe a importância que tem um SGBD no tocante à manutenção dos dados. Imagine como seria se os dados fossem inseridos sem nenhuma garantia e controle. Certamente teríamos um verdadeiro "banco de caos"! Para evitar problemas e dados inconsistentes, há na literatura quatro propriedades que um SGBD deve garantir em uma transação de bancos de dados, a saber as propriedades **ACID** (Atomicidade, Consistência, Isolamento e Durabilidade).

- A **atomicidade** diz respeito à totalidade da transação (atômica), ou seja, a transação ou é efetivada totalmente ou é cancelada, mantendo o estado íntegro anterior. A atomicidade é vista quando no meio de uma transação há uma falha de sistema, onde parte da informação é afetada, tornando dados incompletos.
- A **consistência** é um estado íntegro dos dados que é criado antes de uma transação ser iniciada, onde em caso de falhas este estado é retomado garantindo que o banco de dados esteja em um estado confiável.
- O **isolamento** é visto quando ocorrem duas transações ao mesmo tempo, concorrentemente em uma mesma porção de dados. A transação que iniciou primeiro deve ser encerrada sem interferência da segunda transação, de forma

que a segunda deve esperar o resultado da primeira para assim terminar sua intenção. É comum, as vezes, em sistemas com muitos usuários, transações demorarem mais tempo do que o necessário, não devido a alguma incapacidade de processamento, mas devido ao tempo que deve ser respeitado pelas transações concorrentes.

- A **durabilidade** é a característica que um SGBD deve ter em manter os dados **sadios** por longo tempo, **independendo** de erro de sistema e/ou atualização de versões. Os dados devem ser mantidos pelo tempo necessário até a completude do negócio envolvido.

Diante dos conceitos abordados até agora, você tem uma visão melhor sobre os principais conceitos de bancos de dados relacionais. A seguir temos um exemplo de um banco de dados de Vendas, contendo os conceitos que servem para todo e-book, incluindo as consultas na linguagem SQL, a modelagem da segunda competência e a prática na terceira competência.

1.2 Um Exemplo de Banco de Dados

Os dados em um banco de dados relacional precisam ser armazenados necessariamente ocupando células de tabelas, que são as interseções entre colunas (verticais) e linhas (horizontais). Isso parece trivial, levando em consideração toda a explicação vista nos conceitos iniciais. No entanto, um projetista de banco de dados vai se deparar com situações em um ambiente envolvendo mais de uma entidade e com relações variadas. Embora isso exija um conhecimento melhor acerca de modelagem de bancos de dados (veremos na próxima competência), o projetista deve saber que os dados ficam armazenados em simples tabelas bidimensionais, e toda e qualquer variação de relacionamento e quantidade de tabelas devem se adequar a essa estrutura.

Para fins de exemplo de um banco de dados, criamos um conjunto de tabelas para um sistema de vendas de produtos. Chamaremos de banco de dados de “Vendas”, contendo as tabelas: “Cliente”, “Telefone”, “Endereco”, “Produto”, “Vendedor”, “Loja”, “Venda” e “Venda_Produto”. É possível inferir que uma venda precisa ter: um produto vendido, um vendedor e uma loja. Neste sentido você entende que as tabelas do banco de dados “Vendas” estão de alguma forma

relacionadas entre si e expressam uma situação do mundo real, envolvendo várias entidades. Na Figura 5 verificam-se essas tabelas com alguns dados de exemplo.

O banco de dados “Vendas” armazena dados de um fluxo simples de venda de produtos em um ponto comercial de uma rede de lojas. De forma resumida, um cliente cadastra seus dados, escolhe um produto sugerido por um vendedor em uma das lojas da rede e efetua a compra. O banco de dados armazena os dados da transação que contém a data e hora, os produtos, o valor, o vendedor que fez a venda (para fins de comissão), dados da loja que efetuou a venda (para fins de relatórios gerenciais), o cliente que efetuou a compra (para fins de mala direta e *marketing*) e a quantidade vendida (para fins de controle de estoque).

Cada tabela em um banco de dados tem seu papel fundamental para representar a entidade, tanto no momento da transação como também para tomada de decisões e relatórios. No banco de dados de exemplo visto na Figura 5 (Vendas), a tabela Cliente armazena os dados básicos dos clientes cadastrados. Esta tabela é fundamental para criar um conjunto de dados que podem ser usados para propor, por exemplo, futuras promoções pela equipe de *marketing* da empresa a partir de análises de compra. Por exemplo, é possível saber quais os clientes que mais compram em um determinado mês. Como também saber quais são os clientes que compram o produto “X”, qual quantidade e a periodicidade.

Cliente

PKCPF	Nome	Email
889.458.665-90	Fulano da Silva Xavier	fulano@gmail.com
005.884.632-52	Beltrano Belomonte	beltrano@hotmail.com
448.997.564-23	Cicrano Cunha	cicrano@bol.com
548.654.789-53	Ciriguela Pacheco	ciriguela@gmail.com
544.777.889-65	Fafinha Basctos	rafinha@ig.com.br

Telefone

PKTELEFONE	CPF	Telefone
1	889.458.665-90	81996558877
2	889.458.665-90	81998877788
3	005.884.632-52	21996589632
4	448.997.564-23	85995878965
5	448.997.564-23	85995879654
6	548.654.789-53	11998892314
7	544.777.889-65	12995478246

Endereco

PKENDERECO	CPF	Rua_Numero	Bairro	Municipio	Estado	CEP
1	889.458.665-90	Rua 3, 21	Dois Unidos	Olinda	PE	54269-328
2	889.458.665-90	Rua João Barros, 3	Centro	Recife	PE	50458-588
3	005.884.632-52	Av Abreu Lima, 5	Arruda	Recife	PE	52478-522
4	548.654.789-53	Rua Ave Fátima, 65	Alto do Mandú	Jaboatão	PE	53855-699
5	448.997.564-23	Av Lulia da Silva, 13	Casa Grande	João Pessoa	PB	51588-586

Produto

PKPRODUTO	Descricao	Qtd_Estoque	Unidade_Armazenada	FKLOJA	Valor_Unidade
15486	Amaciante OMU	215	Peça	25	12,00
54896	Sabonete DUVI	512	Pacote C 10	25	31,50
56845	Arroz Tio José	310	Saca C 50Kg	32	59,90
96548	Fubá Vitamío	502	Saca C 20Kg	37	25,60

Vendedor

PKCPF	Nome	FKLOJA
887.545.695-23	José Maria	25
221.565.869-33	Maria da Silva	37
558.669.895-96	Miguel Sarmento	32
441.578.521-52	Benedito de Paula	25

Loja

PKLOJA	Descricao
25	Loja Centro
32	Loja Olinda
37	Loja Paulista
40	Loja João Pessoa

Venda

PKVENDA	Data_Hora	Valor	FKCLIENTE	FKLOJA	FKVENDEDOR
5468	11/04/2019 08:25:41	114,00	889.458.665-90	32	558.669.895-96
5469	10/04/2019 15:32:17	53,15	005.884.632-52	37	221.565.869-33
5470	10/04/2019 14:12:31	84,75	548.654.789-53	25	887.545.695-23
5471	09/04/2019 10:30:25	25,50	448.997.564-23	25	441.578.521-52

Venda_Produto

FKVENDA	FKPRODUTO	Quantidade
5468	56845	1
5468	54896	1
5469	15486	1
5470	56896	3

Figura 5 - Banco de dados Vendas.Fonte: O Autor (2019).
Descrição: A figura possui um conjunto de tabelas formando um banco de dados de sistema de vendas. A tabela Cliente possui alguns registros contendo os atributos: PKCPF, Nome e Email. A tabela Telefone possui registros com os atributos: PKTelefone, CPF e Telefone. A tabela Endereço possui PKEndereço, CPF, Rua_Numero, Bairro, Município, Estado e CEP. A tabela Produto possui registros com os atributos: PKProduto, Descrição, Qtd_Estoque, Unidade_Armazenada, FKLoja e Valor_Unidade. A tabela Vendedor possui registros com os atributos: PKCPF, Nome e FKLoja. A tabela Loja possui registros com os atributos: PKLoja e Descrição. A tabela Venda possui registros com os atributos: PKVenda, Data_Hora, Valor, FKCliente, FKLoja e FKVendedor. A tabela Venda_Produto possui registros com os atributos: FKVenda, FKProduto e Quantidade.

Cada cliente do banco de dados Vendas pode ter um ou mais telefones. Para representar isso, foi criada a tabela Telefone contendo um identificador sequencial (PKTelefone), o CPF do cliente e o telefone. Talvez você se pergunte: "*Esta tabela de telefone poderia ficar na própria tabela de clientes?*". A resposta é sim. Contudo, como já foi dito, entenda que só existem colunas e linhas em

uma estrutura de tabelas. Por isso, caso o cliente tivesse mais de um telefone, a tabela teria algo do tipo: telefone1, telefone2, telefoneN… (explicado na Figura 4). Devido esta limitação entre relacionamento entre tabelas, é criado então uma outra tabela para armazenar os dados da entidade que tem mais de um registro, que neste caso é o telefone. Da mesma forma é a relação entre a tabela Cliente e Endereço. Cada cliente pode ter um ou mais endereços (levando em consideração apenas que um endereço não tenha outro cliente). A tabela Cliente se conecta às tabelas de Telefone e Endereço por meio da chave CPF.

Ainda no banco de dados de Vendas mostrado na Figura 5, é possível verificar que um produto possui como atributos: um identificador (PKProduto), descrição, a quantidade em estoque, unidade armazenada, identificador de loja (FKLoja) e o valor de cada produto. Caso haja necessidade, é possível adicionar outros atributos na tabela Produto com o objetivo de adquirir melhores relatórios. Por exemplo: fornecedores, preço de compra, margem de lucro e melhor temporada para ser vendido. Observa-se também a quantidade de cada produto (QTD_Estoque). Este valor de atributo é subtraído a cada venda que contenha este produto. Por exemplo, ao vender 10 pacotes de algum produto é necessário que haja uma operação no banco de dados que diminua em 10 o valor do estoque atual. Já o atributo Valor_Unidade está diretamente relacionado com o valor pago pelo cliente. O valor total de uma venda é dado pela soma da quantidade vendida de produtos multiplicada pelo respectivo valor unitário, cujo entendimento pode ser facilitado pela seguinte fórmula: SOMA(Quantidade_Vendida_Produto*Valor_Produto). O atributo FKLoja define qual a loja física possui determinado produto. Para fins de relatórios, é possível saber, por exemplo, quantos produtos determinada loja vendeu em um determinado período do ano (aqui envolve a tabela Venda).

Diante do que foi visto com respeito às tabelas Cliente e Produto, fica simples saber os papéis das tabelas Vendedor e Loja da Figura 5. No entanto, as tabelas de Venda e Produto possuem uma relação de muitos para muitos entre elas (conhecido como NxN). Ou seja, **cada venda possui um ou vários produtos e, cada produto pode estar em uma ou várias vendas**. Para representar este tipo de relação é necessário que seja criada uma tabela auxiliar contento ambas as chaves primárias das tabelas envolvidas (tabela Venda_Produto da Figura 5). Sendo assim, esta tabela contém a chave primária da tabela Venda, a chave primária do produto e a sua quantidade. A chave de venda (FKVenda) se repete na medida em que produtos são adicionados. Uma melhor explicação sobre relacionamentos será vista na competência 2.

Antes de iniciarmos efetivamente os conceitos da linguagem SQL, caso você queira acompanhar executando os comandos em um SGBD, você deve instalar o SqLite ou o PostgreSQL em seu computador; e para isso, vá até a Competência 3, no subtópico 3.1 Instalação de um SGBD e Criação do Banco de Dados e siga os passos lá descritos.

1.3 Structured Query Language (SQL)

Com o banco de dados já projetado (ver Figura 5), faz-se necessário um conjunto de instruções para definição, manipulação e controle destes dados em um SGBD: a linguagem SQL (*Structured Query Language* – Linguagem Estruturada de Consulta), que é a linguagem padrão dos SGBD´s relacionais comerciais. O SQL possui um conjunto básico de instruções que é composto por dois tipos de linguagens (ou sublinguagens): linguagem de definição de dados: DDL - *Data Definition Language* (CREATE, DROP e ALTER) e linguagem de manipulação de dados: DML - *Data Manipulation Language* (SELECT, UPDATE, DELETE e INSERT). Outras siglas de sublinguagens podem ser vistas na literatura: linguagem de controle de dados (*Data Control Language* – DCL), linguagem de transação de dados (*Data Transaction Language* – DTL), linguagem de definição de visões (*View Definition Language* - VDL), linguagem de definição de armazenamento (*Storage Definition Language* - SDL) e linguagem de consulta de dados (Data Query Language - DQL). Sugiro você buscar saber mais sobre os principais comandos destas sublinguagens. Adotaremos as linguagens SQL-DML e SQL-DDL.

O primeiro comando SQL é o CREATE. Com este comando é possível criar tabelas e outros objetos em um banco de dados. O formato básico deste comando é: CREATE TABLE NOME_TABELA (NOME_COLUNA TIPO_DADO), onde os termos CREATE e TABLE são palavras reservadas do SQL, NOME_TABELA é o nome da tabela que será criada, e NOME_COLUNA é o nome do atributo. Já o termo TIPO_DADO é o tipo propriamente dito que a coluna representa, podendo ser numérico (INT, FLOAT e DECIMAL), caractere (CHAR e VARCHAR), booleano (TRUE e FALSE), date e tempo (DATE, DATETIME, TIME e TIMESTAMP) e intervalo (INTERVAL). Com exceção dos tipos de dados especiais (e.g. Geográficos), outros tipos de dados podem ser encontrados na literatura, sendo contudo, variações destes citados. Os comandos CREATE a seguir são referentes às tabelas CLIENTE e TELEFONE do banco de dados Vendas mostrado na Figura 5.

```sql
CREATE TABLE CLIENTE (
    PKCPF VARCHAR(15) PRIMARY KEY,
    NOME VARCHAR(50) NOT NULL,
    EMAIL VARCHAR(20)
    );
CREATE TABLE TELEFONE(
    PKTELEFONE INTEGER PRIMARY KEY,
    CPF VARCHAR(15),
    TELEFONE BIGINT NOT NULL,
    FOREIGN KEY (CPF) REFERENCES CLIENTE (PKCPF)
    );
```

Figura 6 - Exemplos de comandos SQL - CREATE.
Fonte: O Autor (2019).
Descrição: A figura mostra dois comandos SQL – CREATE TABLE. O primeiro comando cria a tabela CLIENTE do banco de dados VENDAS mostrado na Figura 5. O segundo comando cria a tabela TELEFONE do mesmo banco de dados. No primeiro comando CREATE é criada a tabela Cliente, tabela esta que possui três atributos: PKCPF, NOME e EMAIL. No segundo comando CREATE é criada a tabela Telefone, tabela esta que possui três atributos: PKTELEFONE, CPF e TELEFONE. A coluna CPF da tabela TELEFONE representa uma chave estrangeira da tabela CLIENTE.

A Figura 6 mostra dois exemplos de comandos de criação de tabelas: SQL – CREATE TABLE. Estes comandos representam a criação de duas tabelas do banco de dados “Vendas” mostrado na Figura 5. O primeiro comando cria a tabela CLIENTE contendo três atributos do tipo VARCHAR: PKCPF, NOME e EMAIL. O atributo PKCPF é a chave primária da tabela, cuja característica é especificada pela cláusula PRIMARY KEY. O atributo NOME não pode ser omitido no momento da inserção dos dados, característica especificada pela cláusula NOT NULL. O valor numérico entre parênteses é o tamanho do dado que o atributo pode receber. Por exemplo, o atributo NOME pode receber até cinquenta caracteres. Já o segundo comando, cria-se a tabela TELEFONE contendo três atributos do tipo INTEGER, VARCHAR e BIGINT, respectivamente: PKTELEFONE, CPF e TELEFONE. O comando FOREIGN KEY informa que o campo CPF da tabela TELEFONE é uma chave estrangeira e faz referência - REFERENCES – com a tabela CLIENTE pela chave primária PKCPF. Outra observação é com respeito à sequência na criação das tabelas. Uma vez que a tabela TELEFONE precisa de um CPF, a tabela CLIENTE deve ser criada e populada primeiro. Em outras palavras, é necessário que haja previamente um registro na tabela CLIENTE cuja chave primária deve ser especificada no momento da inserção na tabela TELEFONE. A cláusula de chave estrangeira (FOREIGN KEY - REFERENCES) pode ser omitida no momento da criação. No entanto, caso não seja especificada, a tabela TELEFONE poderá conter registros sem o seu respectivo dono, ou seja, sem uma referência. Ao especificar o

comando FOREIGN KEY o usuário está informando que só existe telefone com um cliente associado, do contrário haverá um erro ao inserir um registro.

Para que os dados sejam inseridos nas tabelas criadas, é preciso utilizar um comando de inserção de forma que respeite os tipos de dados e as restrições dos comandos SQL - CREATE vistos anteriormente na Figura 6. Para isso, utiliza-se o comando SQL – INSERT INTO, acompanhado da cláusula VALUES. O formato básico deste comando é: "INSERT INTO TABELA VALUES (VALOR1, VALOR2, VALORN)", onde TABELA é o nome da tabela na qual os dados serão inseridos, e os termos VALOR são os dados que serão persistidos em cada coluna, respeitando cada valor o tipo definido no momento da criação da tabela. Por exemplo, os dados que serão inseridos na coluna NOME da tabela CLIENTE devem ser do tipo VARCHAR com até cinquenta caracteres alfanuméricos. Caso seja inserido, por exemplo, um valor do tipo INTEGER na coluna NOME, o SGBD não permitirá que a operação seja realizada. Nas Figuras 7 e 8 verificam-se alguns comandos de inserção SQL – INSERT tomando como base as tabelas dos comandos de criação mostrados na Figura 6: CLIENTE e TELEFONE.

```
INSERT INTO CLIENTE VALUES ('008.845.966-52', 'JOÃO DA COSTA', 'JOAODACOSTA@GML.COM');
INSERT INTO CLIENTE VALUES ('885.665.965-78', 'FULANO DA SILVA', 'FULANOSILVA@GML.COM');
INSERT INTO CLIENTE VALUES ('554.854.547-85', 'JOSÉ DAS CANDOCA', 'DASCANDOCA@GML.COM');
INSERT INTO CLIENTE VALUES ('552.312.689-96', 'MARIA DO ROSÁRIO', 'MARIAROSA@GML.COM');
INSERT INTO CLIENTE VALUES ('785.645.589-21', 'FÁTIMA BERNARDO', 'FATIMABERNA@HOT.COM');
INSERT INTO CLIENTE VALUES ('548.552.332-90', 'MARIA DO BAIRRO');
```

Figura 7 - Exemplos de comandos SQL – INSERT na tabela CLIENTE.
Fonte: O Autor (2019)
Descrição: A figura mostra seis exemplos de comandos SQL-INSERT na tabela CLIENTE, contendo alguns valores fictícios. O último comando (em destaque) é omitida a coluna EMAIL.

```
INSERT INTO TELEFONE VALUES (1, '008.845.966-52', 819965588**);
INSERT INTO TELEFONE VALUES (2, '885.665.965-78', 819965544**);
INSERT INTO TELEFONE VALUES (3, '554.854.547-85', 819978855**);
INSERT INTO TELEFONE VALUES (4, '554.854.547-85', 219954896**);
INSERT INTO TELEFONE VALUES (5, '552.312.689-96', 11213562**);
INSERT INTO TELEFONE VALUES (6, '785.645.589-21', 85998745**);
INSERT INTO TELEFONE VALUES (7, '885.546.325-63', 819963255**);
```

Figura 8 - Exemplos de comandos SQL – INSERT na tabela TELEFONE.
Fonte: O Autor (2019)
Descrição: A figura mostra sete exemplos de comandos SQL-INSERT na tabela TELEFONE, contendo alguns valores fictícios. Um dos comandos (em destaque) provoca um erro no momento da execução, devido ao valor de chave estrangeira não está inserido na tabela de origem como chave primária.

É possível verificar na Figura 7 cinco comandos de inserção **SQL-INSERT** objetivando popular a tabela CLIENTE do banco de dados Vendas mostrado na Figura 5. Observa-se que a tabela CLIENTE possui três atributos do tipo VARCHAR, cada um com uma determinada quantidade baseando-se, de forma coerente, pela particularidade do dado. Caso esta quantidade seja ultrapassada no momento de execução, é exibido um erro pelo SGBD e a operação não é finalizada com sucesso. Os dados recebem aspas simples, pois são necessárias em dados do tipo VARCHAR. Verifica-se em destaque um comando se inserção omitindo uma das colunas. Em SQL-INSERT é possível também inserir dados em apenas algumas colunas, de forma que as colunas omitidas não recebam dado algum, mais conhecido como dado do tipo NULL. Nestes casos, tais colunas não podem estar configuradas como NOT NULL no momento de sua criação (observe a coluna EMAIL da tabela CLIENTE na Figura 6). Para isso, no momento do comando de inserção, é preciso deixar explícito quais colunas receberão dados e omitir aquelas que ficarão nulas (observe o destaque da Figura 7). Esta prática, embora permitida em banco de dados, não é aceita com naturalidade, uma vez que desvaloriza o conjunto de dados para futuras análises e também impacta na necessidade de tratamentos em rotinas de softwares de consultas.

Na Figura 8, é possível verificar sete comandos de inserção SQL-INSERT que tem o objetivo de popular a tabela TELEFONE. A primeira coluna da tabela TELEFONE é do tipo INTEGER, cujos dados são inseridos de forma sequencial. Alguns SGBD´s possuem um tipo de dado que se incrementa automaticamente (isto é, o primeiro registro recebe o número 1, o segundo recebe o número 2 e o enésimo recebe o N em sequência): (MySQL: AUTO_INCREMENT; PostgreSQL: SERIAL; SQL Server: IDENTITY), facilitando a vida do usuário de banco de dados, uma vez que não é preciso saber qual foi o último valor do registro inserido para continuar a sequência. Nestes casos, o nome da coluna “autoincrementada” é omitido do comando **SQL-INSERT**. No SGBD Oracle, a função de incrementar automaticamente é obtida por meio de um objeto de banco de dados chamado SEQUENCE. É possível também verificar na Figura 8 um comando SQL – INSERT (em destaque) que causa um erro pela falta de chave estrangeira referenciando ambas tabelas (levando em consideração apenas os registros inseridos na tabela CLIENTE da Figura 7). Ao tentar inserir este registro (em destaque) no banco de dados, o SGBD informa que este CPF não existe na tabela CLIENTE e impede que a operação seja realizada, uma vez que foi definida uma restrição de referência no comando CREATE TABLE da Figura 6.

Com os dados já inseridos em uma tabela no SGBD, podemos então efetuar consultas nesse BD. Para isso, o comando a ser utilizado é o SQL-SELECT e suas variações. A forma básica deste comando é: SELECT COLUNA1, COLUNA2, COLUNAN FROM TABELA. Onde o SELECT e FROM são palavras reservadas do SQL. As colunas (COLUNA1, COLUNA2, COLUNAN) são os atributos da tabela que serão exibidos na consulta, podendo o usuário definir qual atributo ele quer que retorne. Caso o usuário queira retornar todos os atributos, pode-se utilizar um asterisco "*" no lugar das colunas. Se o usuário efetuar um comando de consulta sem nenhuma condição, que é a forma básica, retornarão todos os registros da tabela consultada. Se esta tabela conter milhões de registros, a consulta será custosa e desnecessária, caso o usuário queira apenas alguns registros, não todos. Para restringir a consulta de forma a retornar apenas os registros necessários e evitar um desperdício de processamento e tempo, utiliza-se a cláusula WHERE contendo alguma condição. Na Figura 9 temos exemplos de consultas às tabelas recém-criadas e populacionadas que foram mostradas nas Figuras 6, 7 e 8.

```
1 SELECT * FROM TELEFONE;
2 SELECT NOME, EMAIL FROM CLIENTE;
3 SELECT CPF, TELEFONE FROM TELEFONE;
4 SELECT * FROM CLIENTE WHERE NOME LIKE 'MARIA%';
5 SELECT * FROM TELEFONE WHERE CPF = '008.845.966-52';
6 SELECT COUNT(*) FROM CLIENTE;
7 SELECT C.NOME, C.EMAIL, T.CPF, T.TELEFONE FROM CLIENTE C
    INNER JOIN TELEFONE T
    ON T.CPF = C.PKCPF;
```

Figura 9 - Exemplos de comandos SQL-SELECTFonte: O Autor (2019)
Descrição: A figura mostra sete comandos SQL-SELECT. O primeiro comando é a forma mais básica. O segundo exibe apenas dois atributos da tabela CLIENTE. O terceiro mostra apenas dois atributos da tabela TELEFONE. O quarto mostra uma condição com operador LIKE. O quinto mostra uma condição de comparação com o operador IGUAL. No sexto comando é mostrada a função de contagem COUNT na tabela CLIENTE. O sétimo e último, mostra um comando mais trabalhado, envolvendo uma junção entre duas tabelas pela cláusula INNER JOIN.

Na Figura 9 é possível verificar sete comandos SQL-SELECT numerados de 1 a 7. O primeiro comando apresenta a forma mais básica do SQL-SELECT. Neste exemplo são retornados todos os registros da tabela TELEFONE. No segundo comando, também são retornados todos os registros da tabela CLIENTE, contudo, apenas dois atributos foram especificados pelo usuário: NOME e EMAIL, ficando de fora o atributo PKCPF. O mesmo é visto no comando da terceira linha, onde são exibidos os atributos CPF e TELEFONE, ficando de fora o atributo PKTELEFONE da tabela TELEFONE. Na quarta

linha o comando faz uso da cláusula WHERE, informando uma condição a ser respeitada. Neste caso, a condição é que o atributo NOME deve conter “MARIA” no início da *string*. Observe que há um sinal de porcentagem no final do termo a ser pesquisado (“%”), que neste caso informa que a *string* do banco de dados deve iniciar com o termo “MARIA”, não importando o que vem após o termo (exemplo, “MARIA DO ROSÁRIO”). Caso este sinal esteja antes do termo (ou seja, “%MARIA”), significa que a *string* do BD deve ser encerrada como “MARIA” (exemplo, “JOSÉ MARIA”), não importando o que vem antes do termo. Caso o sinal de porcentagem esteja em ambos os lados do termo (ou seja,“%MARIA%”), significa que a *string* do BD deve conter o termo, não importando se este esteja no início, no meio ou no final da *string* (exemplo, “ALINE MARIA DA SILVA”). O quinto comando SQL-SELECT da Figura 9 é um exemplo de condição utilizando a cláusula WHERE junto ao operador de comparação IGUAL (“=”). Neste exemplo serão retornados os telefones do CPF: “008.845.966-52”. O comando da linha seis trata-se de um exemplo de uma função de agregação (COUNT) que retorna a quantidade de registros da tabela CLIENTE. O próximo e último comando da Figura 9 trata-se de uma junção entre tabelas (INNER JOIN - ON) e o uso de ALIAS. As palavras reservadas INNER JOIN - ON são usadas para especificar um relacionamento entre duas tabelas. No exemplo da Figura 9, as tabelas CLIENTE e TELEFONE são relacionadas pelo atributo em comum CPF (ON CPF = PKCPF). Caso existam mais tabelas para serem relacionadas, o usuário deve fazer uso de mais de um INNER JOIN – ON, isto é, para cada dupla de tabelas relacionadas deve haver um comando de junção. Por fim, o ALIAS (representada pela declaração AS no SQL), que é um nome dado temporariamente para tabelas e atributos, é visto em “...CLIENTE C” e em “...TELEFONE T”. Este artifício é usado apenas para o usuário não ter que reescrever o nome completo da tabela quando deve juntar logo abaixo após o termo ON. Do contrário, deveria ser “TELEFONE.CPF = CLIENTE.PKCPF”. É mais cômodo utilizar: “T.CPF = C.PKCPF” e também nos atributos “C.NOME. T.TELEFONE, ...”. O ALIAS pode ser usado também em atributos para substituir o nome original da coluna e permitir um nome mais amigável. Por exemplo, digamos que o atributo "cliente' que ser modificado para "cliente _id". Então, o comando deve ficar: “SELECT cliente AS cliente_id FROM NOME_DA_TABELA.

Funções de agregação retornam um valor numérico e são usadas como ferramentas de análises, como por exemplo: Média, Máximo, Mínimo e Soma. Na terceira competência abordaremos mais a fundo estas funções e conceitos.

O próximo comando SQL é o **UPDATE**. Este comando tem o objetivo de atualizar algum dado na tabela. Por exemplo, em casos onde o usuário digite o dado errado e o insira, mesmo assim, no banco de dados. Ou, também, em casos que um registro foi inserido no BD com algum atributo faltando e o usuário precisa alterá-lo. A forma básica deste comando é escrita da seguinte forma: UPDATE TABELA SET COLUNA1 = DADONOVO, COLUNA2 = DADONOVO2 WHERE COLUNACONDIÇÃO = CONDIÇÃO. Onde UPDATE, SET e WHERE são palavras reservadas do SQL. As colunas (COLUNA1, COLUNA2) são os atributos que serão atualizados. Os dados novos (DADONOVO1, DADONOVO2) são os dados que serão inseridos no lugar dos atuais. COLUNACONDIÇÃO é um atributo chave que deve satisfazer a condição (CONDIÇÃO) definida pelo usuário. Na Figura 10 é possível verificar dois exemplos do comando SQL-UPDATE.

```
1 UPDATE CLIENTE SET EMAIL = 'MARIAROSA@HOT.COM'
    WHERE PKCPF = '548.552.332-90';
2 UPDATE TELEFONE SET TELEFONE = '81986254488'
    WHERE PKTELEFONE = 4;
```

Figura 10 - Comando SQL UPDATE.Fonte: O Autor (2019)
Descrição: Nesta figura são exibidos dois comandos SQL-UPDATE. O primeiro altera um EMAIL do cliente cujo CPF é definido como condição para a operação. O segundo comando é uma atualização de telefone cuja condição é feita pela chave primária da tabela.

Antes de você executar algum comando SQL-UPDATE, é de extrema importância que você só o execute com a cláusula WHERE junto a uma condição envolvendo apenas um registro (utilize a chave primária) ou um conjunto de registros que, de fato, você queira alterar. Ou seja, se o usuário executar um comando de atualização sem especificar uma condição, todos os registros daquela tabela serão modificados. E acredite, depois que o comando foi executado e, dependendo da configuração do SGBD, estes dados não voltarão atrás como antes, ou seja, não existe CTRL-Z em BD.
Tome muito cuidado!

Na Figura 10 é possível verificar dois comandos de atualização de dados. O primeiro comando realiza uma atualização do e-mail do cliente, cujo CPF é o "548.552.332-90". No segundo comando é alterado o telefone cujo identificador (PKTELEFONE) é o número 4. Você poderia utilizar, neste último comando, o próprio telefone como condição, levando em consideração que nenhum outro cliente possua o mesmo número. Caso o usuário utilize uma condição que seja ambígua (por exemplo, "...WHERE NOME LIKE '%MARIA%'"), mais de um registro será alterado e os dados na tabela serão prejudicados.

O próximo comando SQL é o DELETE. Este comando tem o objetivo de apagar um registro da tabela em um banco de dados. De forma similar ao comando SQL-UPDATE, este comando precisa da cláusula WHERE para restringir o número de registros que serão apagados. Ou seja, não se deve executar o comando DELETE sem especificar uma condição. Tome cuidado!

A forma básica deste comando é especificada da seguinte maneira: **DELETE FROM** TABELA **WHERE** COLUNACHAVE = IDENTIFICADOR. Onde DELETE, FROM e WHERE são palavras reservadas do SQL; TABELA é a tabela que o registro será apagado; COLUNACHAVE é a coluna que deve conter um valor chave para satisfazer a condição; e IDENTIFICADOR é o valor da condição. Na Figura 11 é possível verificar três comandos SQL-DELETE, onde o último dentre os três contém um erro quando executado.

```
1 DELETE FROM CLIENTE
    WHERE PKCPF = '548.552.332-90'
2 DELETE FROM TELEFONE
    WHERE TELEFONE = '81996558899'
3 DELETE FROM CLIENTE
    WHERE PKCPF = '885.665.965-78'
```

Figura 11- Comandos SQL DELETE.
Fonte: O Autor (2019).
Descrição: A figura contém três comandos SQL DELETE. O primeiro comando utiliza a chave primária da tabela CLIENTE para excluir um registro. O segundo comando exclui um registro da tabela TELEFONE especificando o próprio telefone como condição. O terceiro e último comando tenta excluir um registro da tabela CLIENTE, contudo, este comando não é executado devido a uma regra que verifica referências entre tabelas.

A Figura 11 contém três comandos SQL-DELETE. O primeiro exclui um registro da tabela CLIENTE especificando qual cliente deverá ser excluído, utilizando seu CPF como condição. No segundo comando é excluído um TELEFONE tendo como parâmetro de condição o próprio telefone. Neste exemplo, a utilização de uma chave como o CPF não seria interessante, uma vez que o CPF pode conter mais de um telefone e o usuário não queira excluir todos telefones. O terceiro comando (em destaque), é feita uma exclusão a partir do CPF do CLIENTE. No entanto, como o cpf "885.665.965-78" possui um telefone a ele vinculado (veja comandos SQL-INSERT na Figura 8), ou seja, a tabela TELEFONE contém um registro com esse CPF, não será possível realizar esta exclusão, uma vez que há uma referência entre as tabelas CLIENTE e TELEFONE por meio do CPF. Neste caso, o usuário pode excluir primeiramente o registro da tabela TELEFONE, e logo após o registro da tabela CLIENTE. Este comportamento é definido por meio da criação da tabela TELEFONE, quando são especificados os termos: FOREIGN KEY – REFERENCES (ver Figura 6). Caso isso não seja feito, o comando DELETE poderá ser executado e, então, teremos uma tabela de TELEFONE sem clientes relacionados. Este controle de integridade é necessário para manter os dados confiáveis, mais conhecido como controle de integridade referencial.

Para maior aprofundamento confira os **MATERIAIS COMPLEMENTARES** da primeira competência.

Caro estudante, nesta competência vimos os principais conceitos relacionados a banco de dados relacionais com exemplos práticos. Na próxima competência veremos como estes conceitos se relacionam ao de modelagem conceitual de banco de dados, utilizando duas formas de criar esquemas conceituais por meio de ferramenta CASE (Computer-Aided Software Engineering – Engenharia de Software Assistida por Computador): EER (Entidade Relacionamento Estendido) e UML (Linguagem de Modelagem Unificada). Bons estudos!

2.Competência 02 | Elaborar um Modelo de Entidade-Relacionamento

Todo sistema de computação precisa ser projetado de forma a evitar diversos problemas no futuro. Em Bancos de Dados isso não é diferente, uma vez que este armazenará regras do negócio a ser gerido por um sistema utilizando tabelas relacionadas. Neste sentido, esta competência tem o objetivo de fazer você entender os principais conceitos de um projeto conceitual de banco de dados até sua implementação em um SGBD, utilizando ferramentas e linguagens apropriadas.

2.1 Modelagem de Banco de Dados

A modelagem de um banco de dados é um processo que tem como objetivo definir a estrutura e regras de um BD, iniciando-se na interpretação do minimundo (representação de uma parcela do mundo real de interesse do usuário) até a implementação em um SGBD. Este processo possui diferentes níveis de abstração (alto nível de abstração significa que está perto da linguagem humana, e baixo nível de abstração está mais perto da implementação), como por exemplo: nível dos usuários (alto nível, nível externo, sem nenhum termo técnico nem ferramenta CASE); nível conceitual (possui uma representação do negócio em ferramenta CASE, mas não possui tipos de dados); nível lógico (possui uma representação em ferramenta CASE contendo os tipos de dados); e nível físico (define os comandos DDL em um SGBD relacional). A partir do nível lógico, a implementação depende de um SGBD específico. Já os níveis Externo e Conceitual, são independentes de tecnologia. Na Figura 12 é possível verificar os diferentes níveis de abstração no processo de desenvolvimento de um banco de dados de forma ilustrada.

O processo é iniciado pelo entendimento do negócio por meio de um minimundo (descrição das regras do negócio), onde o cliente deve informar como seu negócio funciona. Tomando como exemplo o banco de dados Vendas visto na primeira competência, seu minimundo pode ser: "*Aqui na minha loja, vendemos de tudo um pouco. Vendemos roupas, calçados, acessórios e artigos para presentes. Para compras acima de 200 reais, damos um desconto. Outro desconto é quando o cliente curte e compartilha nossa fanpage no Facebook. Queremos também vender diretamente pela internet, onde hoje o cliente faz o pedido por telefone e o processo ainda é muito manual. Nosso estoque é feito em uma planilha Excel, e sempre quando um cliente compra, temos que dar baixa manualmente. A loja possui vários funcionários, onde damos uma comissão de 10% sobre o valor*

vendido, caso este valor supere à 15 mil reais. Temos 3 filiais nos bairros vizinhos e queremos fazer uma divulgação separada por bairros. A filial do bairro que mais vender vai concorrer a prêmios no final do ano. E o cliente que mais comprar, tem chance dobrada.”.

Após o entendimento do minimundo, é preciso definir quais são os requisitos que o cliente pretende ver funcionando em seu sistema. No caso do minimundo citado, o projetista deve identificar, junto ao cliente, quais as transações e possíveis tabelas que este banco de dados vai comportar. Neste minimundo podemos inferir que este sistema precisará de algumas tabelas básicas, dentre elas, as tabelas vistas na Figura 5 do banco de dados Vendas.

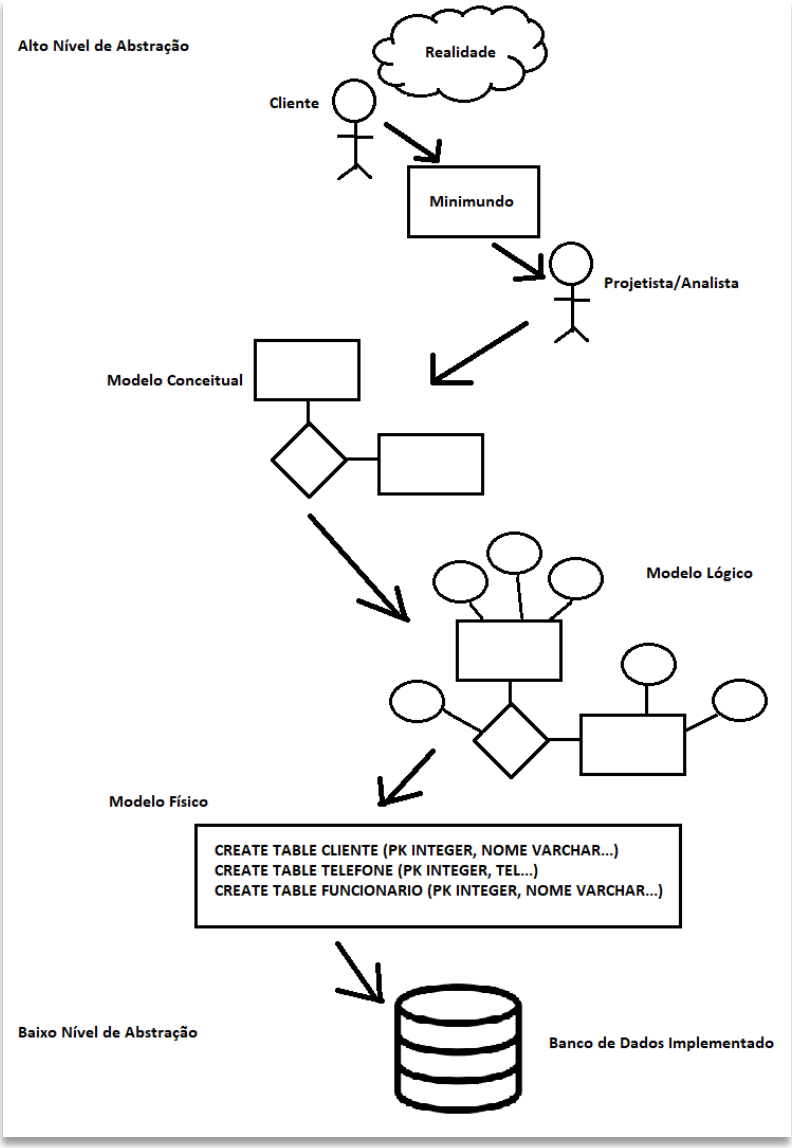

Figura 12 - Níveis de Abstração em projeto de Bancos de Dados.

Fonte: O Autor (2019).
Descrição: Nesta figura verificam-se os diferentes níveis de abstração para um projeto de banco de dados. No primeiro nível o cliente deve informar regras e detalhes do seu negócio por meio de um minimundo para um projetista de banco de dados. Logo após, o projetista interpreta este minimundo e inicia uma modelagem conceitual, omitindo detalhes específicos de dados. No nível lógico, o projetista define tipos de dados e regras do negócio a partir do modelo conceitual anterior. O próximo passo é gerar comandos SQL implementáveis no modelo físico. E, por fim, implementar estes comandos em um SGBD comercial.

A partir deste entendimento do minimundo, são criados Esquemas em linguagens diagramáticas do modelo conceitual pelo projetista, com relacionamentos e regras entre as partes envolvidas no negócio, omitindo detalhes de tipos de dados e complexidade de regras específicas (Figura 12, Modelo Conceitual). No nível lógico é preciso definir os tipos de dados já baseando-se em uma distribuição de SGBD comercial, uma vez que os tipos podem variar de uma distribuição para outra. A diferença entre o nível conceitual e o nível físico está na complexidade. No conceitual há um "cuidado" para não passar para os clientes, donos do negócio, regras, tipos e linguagem técnica. Neste momento ainda há uma interação grande entre o cliente e o projetista. No nível lógico (Figura 12, Modelo Lógico), os artefatos produzidos serão lidos/interpretados por especialistas e profissionais de bancos de dados, e, portanto, podem conter termos mais técnicos. O próximo passo é gerar os comandos SQL-DDL do SGBD (Figura 12, Modelo Físico). Neste momento é necessário já ter uma instância do banco de dados em funcionamento, haja vista que os comandos serão rodados nele. Com todos esses processos terminados, teremos então um banco de dados em pleno funcionamento.

Antes de entrarmos no desenvolvimento do modelo conceitual de banco de dados usando uma linguagem diagramática, é preciso que você entenda o que são os conceitos de: Modelo, Linguagem e Esquema. Segundo os autores Brambilla, Cabot e Wimmer (2012), um **modelo** pode ser conceituado como uma representação simplificada ou parcial da realidade a ser modelada, definido para realizar uma tarefa ou para chegar em comum acordo em um tópico complexo. Já a **linguagem** é um conjunto de símbolos/palavras/termos definidos por regras sintáticas e semânticas, que em conjunto transmitem uma mensagem acerca de algum fato. Por exemplo, a linguagem padrão do país: o Português brasileiro. Nesta linguagem há um conjunto de regras sintáticas e semânticas bem definidas. Por exemplo, em uma frase deve conter um sujeito, o verbo e um complemento: "*O homem comeu a comida*". Estas regras devem ser seguidas conforme estabelecido, do contrário a frase não fará sentido. Por último, temos os **esquemas,** que são os artefatos produzidos pelo projetista no

momento do desenvolvimento, utilizando uma linguagem diagramática em uma ferramenta do tipo CASE para criar um modelo de dados. Na Figura 13 a seguir, temos uma ilustração deste cenário.

Figura 13 - Modelo, Linguagem e Esquemas.
Fonte: O Autor;
Descrição: Nesta figura verificam-se os conceitos de Modelo, Linguagem, e Esquemas.

Tendo estes conceitos entendidos, veremos agora os modelos ER (Entidade-Relacionamento) e EER (Entidade-Relacionamento Estendido), a linguagem UML, e duas ferramentas CASE.

2.2 Modelo ER e ER Estendido

O **modelo ER (Entidade-Relacionamento)** foi proposto por Chen (1976) e é largamente utilizado pela comunidade de banco de dados. Embora seja conhecido como modelo de dados, trata-se de uma linguagem diagramática (linguagem visual), e utilizaremos a palavra “modelo” por convenção. Este modelo recebeu alguns conceitos adicionais por parte dos autores Elmasri e Navathe (2005) se tornando o modelo EER (Entidade-Relacionamento Estendido). O modelo ER possui uma sintaxe e uma semântica como qualquer outra linguagem diagramática, contendo basicamente três construtores: **Entidade (retângulo)**, **Relacionamento (losango)** e **Atributo (elipse)**. As entidades representam algum objeto do mundo real que possua atributo: pessoa, animal, veículo, produto, etc. O relacionamento é a ligação entre entidades: uma pessoa compra produtos. Por fim, temos os atributos, que são características dessas entidades: cor, idade, descrição, país de origem, classificação, etc. Na Figura 14 é possível verificar um exemplo de modelagem de duas entidades relacionadas com seus respectivos atributos.

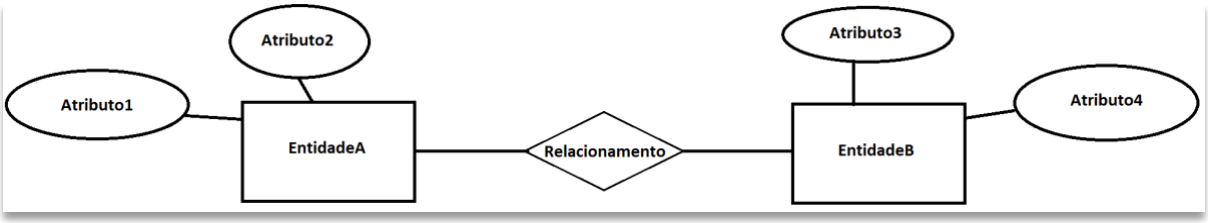

Figura 14 - Exemplo de modelagem utilizando o modelo ER.
Fonte: O Autor (2019).
Descrição: Nesta figura observa-se uma modelagem utilizando o modelo ER. No esquema, estão relacionadas duas entidades com dois atributos cada.

A Figura 14 contém um esquema utilizando o modelo ER, que possui duas entidades (EntidadeA e EntidadeB) contendo dois atributos cada (Atributo1 e Atributo2 para a EntidadeA; e Atributo3 e Atributo4 para EntidadeB), e também um relacionamento entre as entidades. Este relacionamento, por conter apenas duas entidades conectadas, é chamado de relacionamento binário (se fossem três entidades, ternário; quatro entidades, quaternário; e assim por diante). Dificilmente você vai se deparar com uma situação que exija um relacionamento quaternário. Caso aconteça, tente simplificar ao máximo, passando para um ternário acompanhado com um binário.

Tomando como exemplo o banco de dados Vendas visto na primeira competência, podemos criar um exemplo de esquema com as entidades Cliente e Endereço, onde um Cliente tem um ou vários Endereços. Na Figura 15 é possível verificar esta modelagem.

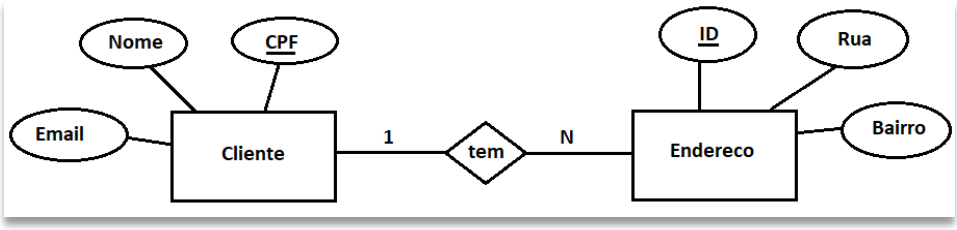

Figura 15 - esquema com as entidades Cliente e Endereço
Fonte: O Autor (2019).
Descrição: Relacionamento Binário, com cardinalidade 1xN, com as entidades Cliente e Endereço; sendo a Cliente contendo os atributos: CPF (Chave primária), Nome e Email. Já a endidade Endereço com os atributos: Identificador (Chavee primária), Rua e Bairro.

Na Figura 15 é possível verificar um esquema contendo duas entidades: Cliente e Endereco. A entidade Cliente possui os atributos CPF (atributo chave), Nome e Email. A entidade Endereco possui como atributos: um Identificador (atributo chave), Rua e Bairro. Há uma relação

entre as entidades ("tem"), sendo interpretado como UM cliente tem N endereços. Estas relações, quando são transformadas para scripts **SQL-DDL, impactam em estruturas de tabelas relacionadas. Neste caso, esta relação de 1 para N, quer dizer que a chave primária do lado 1 servirá de chave estrangeira no lado N. Ou seja, o CPF do Cliente será uma coluna na entidade Endereco (FKCPF, por exemplo). Esta característica de quantidade de instâncias de entidades em um relacionamento tem o nome de Cardinalidade**, e pode variar entre 1x1 (um para um); 1xN (um para muitos) e NxN (muitos para muitos). A seguir, na Figura 16, verifica-se um resumo de todos os construtores do modelo ER, como também dois exemplos contendo os conceitos de Cardinalidade e Participação.

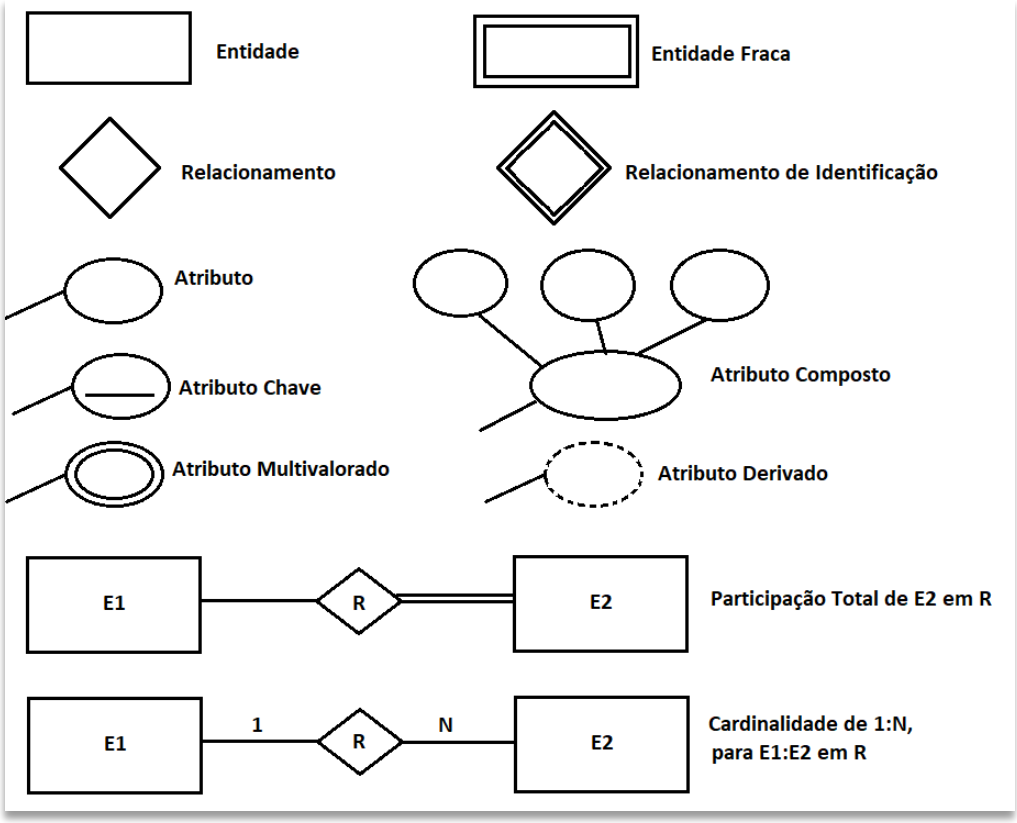

Figura 16 - construtores do modelo ER
Fonte: Elmasri e Navathe (2005).
Descrição: Principais construtores do modelo ER.

Para fins do curso utilizaremos os construtores: Entidade, Relacionamento, Atributo simples, Atributo Multivalorado e Atributo Chave. No entanto, os conceitos de Cardinalidade e Participação são importantes para um bom projeto de banco de dados. **Cardinalidade**, como já

expliquei, está relacionado à quantidade de instâncias envolvidas no relacionamento (1x1; 1xN; NxN). Já o conceito de **Participação** está na obrigatoriedade daquela entidade participar ou não do relacionamento. Por exemplo, na Figura 16 a entidade E2 possui participação TOTAL no relacionamento (representado pelas duas linhas conectando a entidade ao relacionamento), que significa que a entidade E2, para ser cadastrada, deve possuir uma relação com E1. Já o lado da entidade E1, como a participação é parcial (apenas uma linha), a entidade não precisa participar do relacionamento para ser criada. Tomando como exemplo deste conceito, um relacionamento de "1xN" do tipo "Pessoa tem Filhos"; o lado Pessoa não precisa necessariamente ter um filho. Contudo, no lado "Filhos", é uma pré-condição que este tenha um genitor. Neste caso a conexão entre Filhos e o relacionamento será dupla informando a participação total.

Com os principais conceitos do ER abordados, já é possível entender uma proposta completa de modelo para o banco de dados "Vendas" mostrado na Figura 5 da primeira competência. Este modelo é visto a seguir na Figura 17.

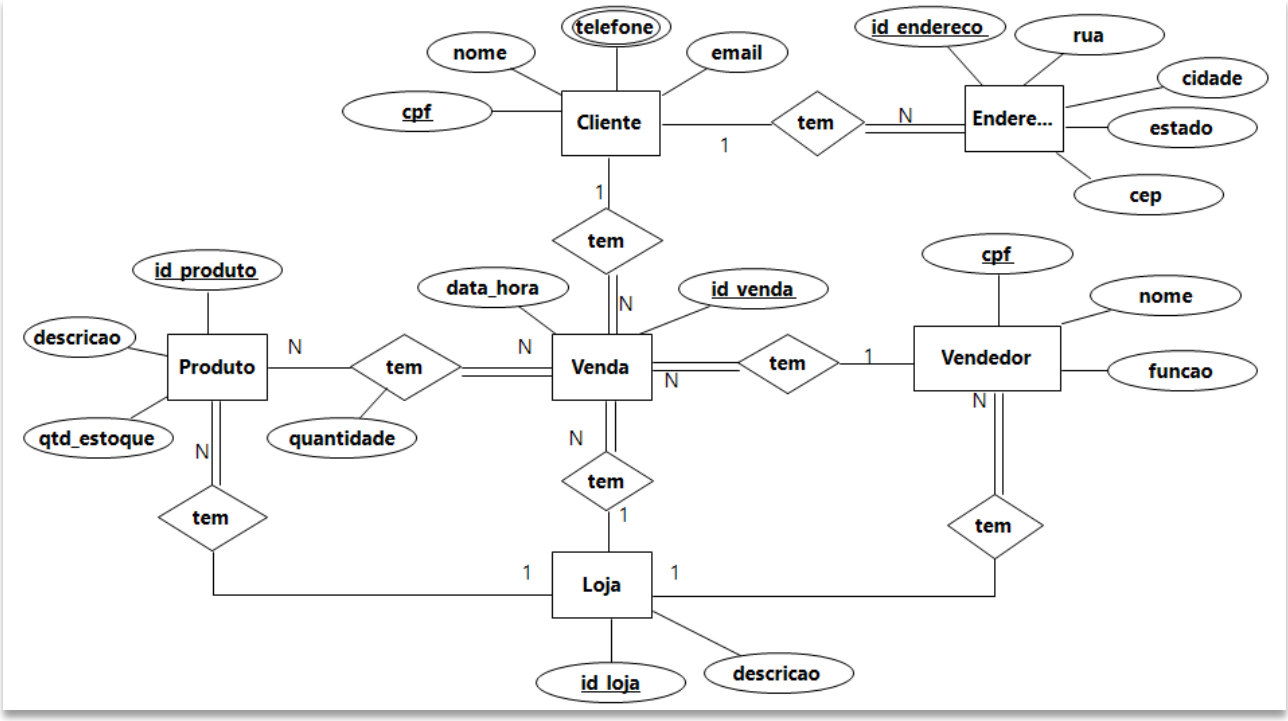

Figura 17 - Modelo de dados para o banco de dados Vendas.
Fonte: O Autor (2019).
Descrição: Nesta figura verificam-se entidades relacionadas, compondo o banco de dados Vendas que será desenvolvido na disciplina.

Na Figura 17 é possível verificar uma proposta completa de um esquema para o banco de dados Vendas visto na Figura 5 da primeira competência. Neste esquema não são encontradas as tabelas de Telefone e Venda_Produto (contidas no BD Vendas). Esta omissão é devido ao fato de: na tabela Telefone o atributo é do tipo Multivalorado e, no caso da tabela Venda_Produto é devido ao fato do relacionamento ter Cardinalidade NxN (muitos para muitos). O primeiro caso, quando o atributo é do tipo Multivalorado (quando pode ter um ou mais registros) é criada uma tabela para representar esta característica, uma vez que só podemos armazenar dados em tabela contendo linhas e colunas. No segundo caso, quando há um relacionamento cuja cardinalidade é NxN, as entidades do relacionamento fornecem suas respectivas chaves primárias para uma terceira tabela. Neste caso, uma Venda pode ter um ou mais produtos e, um produto pode estar em uma ou mais vendas. Sendo assim, a necessidade de criar uma tabela contendo as chaves dos Produtos juntas com as chaves das Vendas. Outro exemplo seria entre as entidades Cliente e Endereco, uma vez que em um endereço pode ter mais de um cliente (um cliente pode ter um ou mais endereços e, um endereço pode ser de um ou mais clientes). No entanto, o relacionamento entre Cliente e Endereco visto na Figura 17 tem Cardinalidade 1xN, assumindo que um endereço tem apenas um cliente, e um cliente pode ter um ou vários endereços.

Como um exercício de fixação, tente identificar os diferentes relacionamentos, entidades, atributos, atributos chaves, atributos multivalorados, cardinalidades e participações contidos no esquema da Figura 17. Tente, por exemplo, inferir que “Uma Loja tem um ou vários Vendedores”, no relacionamento entre as entidades Loja e Vendedor. É importante, também, você observar que não utilizamos acentos nos nomes de entidades (omitimos o cê-cedilha ‘Ç’ na entidade Endereco).

Os conceitos abordados do modelo ER são suficientes para você completar esta disciplina e desenvolver modelos de dados para todo o curso e futuros projetos. Como já entendemos o modelo ER, veremos uma breve descrição da extensão deste modelo: EER (Entidade-Relacionamento Estendido). O **modelo EER** contém construtores mais específicos para representar situações não abordadas pelo ER. Estes construtores formam os conceitos de: Herança (superclasse e subclasse), Especialização e Generalização, e Categoria. O conceito de herança em EER é visto quando você tem uma entidade mais genérica chamada de superclasse (entende-se aqui que uma classe é uma entidade) gerando uma entidade mais específica chamada de subclasse. Um exemplo

de superclasse pode ser uma entidade Veículo, cujos atributos podem ser: Ano, Fabricante, Modelo, Quantidade de Portas, e Tipo Combustível. Como subclasses (ou classes filhas) da entidade Veículo, podemos ter a entidade Caminhão, que vai herdar todos os atributos da classe pai (ou superclasse), adicionando atributos específicos de caminhões: Peso Carga, Quantidade de Eixos, Tamanho da Caçamba, etc. E também uma subclasse Carro, com os atributos: Número de passageiros e Tamanho do porta malas. O conceito de Herança é importante para você evitar repetir nomes de atributos em entidades diferentes. Os atributos em comum (ou genéricos) ficam nas superclasses (entidade/classe pai) e os atributos específicos ficam nas subclasses (entidade/classe filha). Quando são criadas as tabelas em banco de dados, as superclasses fornecem suas chaves primárias para as subclasses, em um relacionamento de Cardinalidade de 1xN, evitando a redundância de dados.

Os conceitos de **Especialização** e **Generalização** são decorrentes da Herança, uma vez que especializar uma classe significa criar uma classe filha partindo da classe pai, com atributos específicos; e generalizar é tornar uma classe que é filha em uma classe pai mais genérica. No exemplo da superclasse Veículo, a especialização desta classe pai é a classe Caminhão; já a generalização é encontrada quando a classe filha (Caminhão) torna-se uma classe mais genérica, ou seja, a classe Veículo. Na especialização, os atributos específicos são inseridos nas classes filhas, já na generalização todos os atributos dos filhos (não repetidos) são inseridos em uma só classe, na superclasse. No exemplo de Veículo, a generalização pega os atributos do Caminhão e do Carro e os coloca em Veículo, criando uma única superclasse, neste caso ficaria: Veículo (Ano, Fabricante, Modelo, Quantidade de Portas, Tipo Combustível, Peso Carga, Quantidade de Eixos, Tamanho da Caçamba, Número de Passageiros, Tamanho do Porta Mala). Observe que, caso um registro fosse de um Carro simples, os atributos referentes ao Caminhão ficariam nulos. Do mesmo jeito seria quando um registro de Caminhão fosse inserido, os atributos do Carro não teriam valores. Agora você entende o porquê utilizar estes conceitos?

O último conceito que abordaremos do modelo EER é o de **Categoria** (ou tipo UNIÃO). Categoria é visto quando uma classe filha (ou subclasse) possui mais de uma classe pai (ou superclasse). Sendo assim, as classes filhas possuem características de mais de uma classe pai. Por exemplo, suponhamos que temos três entidades: Pessoa, Banco e Empresa. Neste caso, um proprietário de um Carro pode ser uma destas entidades. Com isso, podemos criar uma subclasse

chamada Proprietário, e herdar diferentes características. Na Figura 18 é possível verificar um exemplo de Herança (Figura 18A e 18B) e Categoria (Figura 17C) utilizando o modelo EER.

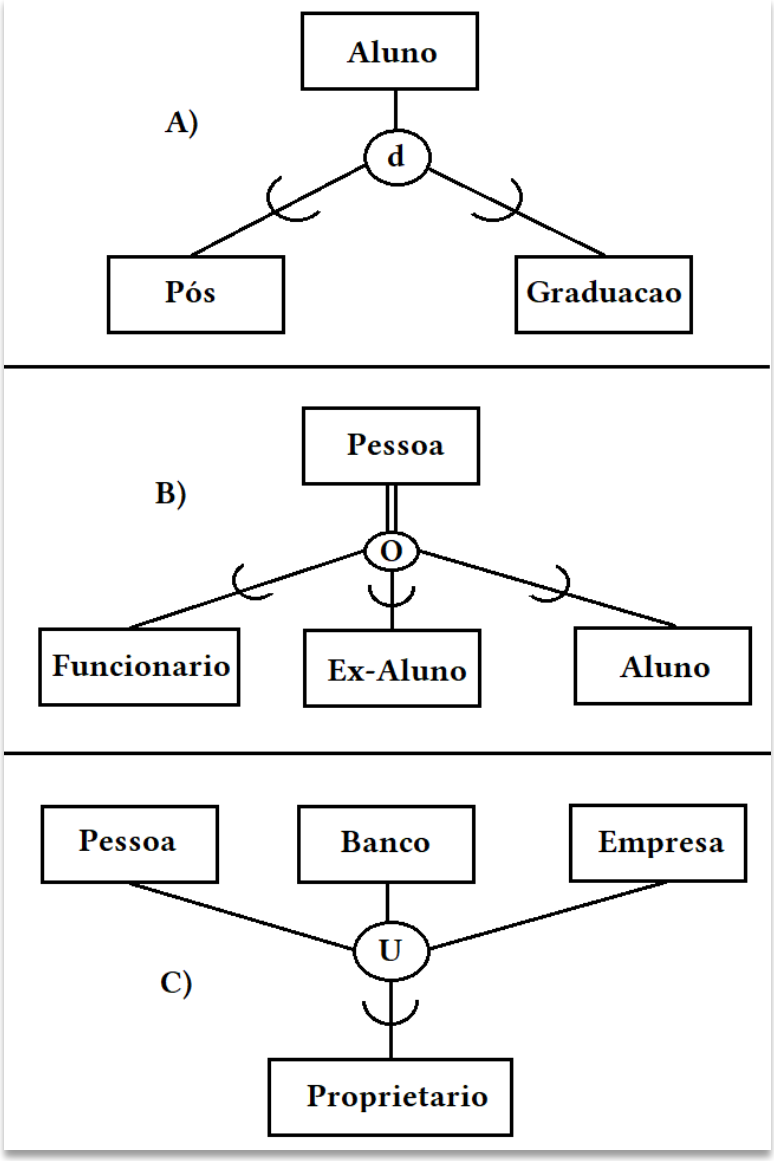

Figura 18 - Conceitos do Modelo ER Estendido.
Fonte: Elmasri & Navathe (2005) Adaptado.
Descrição: Nesta figura verificam-se os conceitos de Herança e Categoria. A figura 'A' mostra uma Herança disjunta, a figura 'B' mostra uma Herança sobreposta, e a figura 'C' mostra uma Categoria tipo União.

Na Figura 18 é possível verificar os conceitos de Herança (Figura 18A e 18B) e o conceito de Categoria/Tipo União (Figura 18C) utilizando a sintaxe do modelo EER. Na Figura 18A, temos uma herança do tipo Disjunta (representada por um "d" dentro do círculo). Disjunta significa

que a Herança é do tipo OU-Exclusivo, que no caso da figura, um Aluno só pode estar matriculado na Graduação ou na Pós, mas nunca nas duas ao mesmo tempo. Já na Figura 18B, a letra "O" dentro do círculo significa "*Overlapping*", ou sobrepostas. Neste caso, as subclasses da superclasse Pessoa podem existir ao mesmo tempo. Ou seja, uma pessoa pode ser Funcionária(o), Ex-Aluna(o) e Aluno simultaneamente. Por último, temos o conceito de Categoria ou Tipo União na Figura 18C. Neste exemplo temos três superclasses (Pessoa, Banco e Empresa) gerando uma subclasse Proprietário. A diferença entre uma possível generalização de uma Herança para uma Categoria está nos diferentes tipos de entidades que a Categoria possui, que no caso da Herança, as subclasses devem ser de um tipo em comum.

Para que você entenda melhor os conceitos do modelo EER, sugiro que faça pesquisa de exemplos de modelagens utilizando Herança e Categoria.

2.3 UML

A UML (*Unified Modeling Language*, ou Linguagem de Modelagem Unificada) é uma linguagem muito utilizada para criar modelos de sistemas em engenharia de software. Contudo, na área de bancos de dados ela também é utilizada e alguns projetistas preferem esta linguagem. Existem alguns trabalhos que comparam a UML e o ER com o objetivo de expor limitações e pontos fortes. De qualquer forma, veremos os principais conceitos da UML para projetos de bancos de dados.

Em um primeiro momento, é preciso que você se familiarize com a sintaxe da UML. Contudo, UML é formada por inúmeros Diagramas: Diagrama de Classe, Diagrama de Objetos, Diagrama de Pacotes, Diagrama de Casos de Uso, etc. Um Diagrama é um conjunto de construtores visuais, estruturados e simplificados para criar uma modelagem em um determinado contexto de processo envolvendo sistema. Por exemplo, o Diagrama de Classe, que é o que utilizaremos nos exemplos, serve para modelar aspectos estruturais dos sistemas, como classes, atributos, métodos e as relações entre estas classes. Já o Diagrama de Casos de Uso é usado para especificar aspectos comportamentais, como funcionalidades do sistema por meio de usuários do produto. Na Figura 19 é possível verificar um exemplo de Diagrama de Classe UML que representa uma forma alternativa ao esquema da Figura 15 modelada em ER.

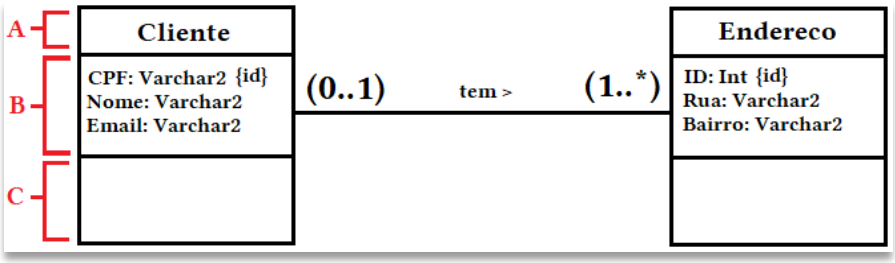

Figura 19 - Relacionamento Binário de cardinalidade 1xN utilizando a linguagem UML.
Fonte: O Autor.
Descrição: Nesta figura verifica-se um relacionamento entre as classes Cliente e Endereço, utilizando a linguagem UML.

Na Figura 19, é possível verificar uma modelagem UML equivalente ao esquema da Figura 15 desenvolvida em ER. Observe que uma classe UML possui três retângulos, um acima do outro, para alocar suas características (em vermelho, os espaços: A, B e C). No primeiro compartimento (espaço "A" na Figura 19), é inserido um nome para a classe. Este nome será o mesmo nome da tabela em um banco de dados. No segundo compartimento (espaço "B" na Figura 19), são inseridos os atributos da classe. Estes atributos serão as colunas da tabela em banco de dados. No último compartimento (espaço "C" na Figura 19), são inseridos métodos (ou alguma ação) da classe. Em programação Orientada a Objetos, métodos são rotinas que determinado objeto vai realizar. Por exemplo, um objeto do tipo Pessoa, poderia ter como métodos: Andar, Correr, Cantar e Passear. Já um objeto do tipo Carro, poderia ter como métodos: Buzinar, Frear, Marcha Ré e Acelerar. Como o exemplo da Figura 19 é uma equivalência ao esquema da Figura 15 (em ER não existem métodos), não inserimos nenhum método. Contudo, neste caso, poderíamos sugerir, para a classe Cliente, os métodos: Tipo Cliente (saber se cliente é bom ou mal pagador), Pegar CPF, Classificar Cliente, etc. Outro ponto específico do Diagrama de Classes UML são os conceitos de Cardinalidade e Participação, vistos no ER. Em UML, estes conceitos podem ser vistos na Figura 19 pelos números entre parênteses: "(0..1) e (1..*)". O primeiro parêntese "(0..1)" significa que não é obrigatória a participação da classe Cliente no relacionamento (representado pelo número zero) e quando a classe participa, pode ter apenas um. Já o segundo parêntese "(1..*)" informa que é preciso que a classe Endereco participe do relacionamento (representado pelo número 1), e o asterisco informa que podem participar inúmeras instâncias desta classe. Em resumo, este relacionamento é do tipo 1xN, representado pela notação UML. O conceito de obrigatoriedade é explícito na notação UML (se estiver com o número 1). Já utilizando o ER, este conceito é representado por uma linha dupla entre a entidade e o relacionamento.

Como atividade objetivando absorver o conteúdo dos conceitos da UML, você pode sugerir um esquema equivalente ao banco de dados “Vendas” desenvolvido em ER, mostrado na Figura 17, utilizando o Diagrama de Classe da UML.

2.4 Ferramenta CASE – EERCASE

Ferramentas do tipo CASE (*Computer Aided Software Engineering*, ou Engenharia de Software Assistida por Computador) são utilizadas para auxiliar o desenvolvimento de sistemas e bancos de dados. O processo se inicia com uma modelagem do sistema a ser desenvolvido, como entidades, relacionamentos, atributos e tipos de dados; e logo após são gerados códigos para implementação deste sistema. Caso seja um sistema de software, são geradas classes, métodos e atributos para ele. Caso seja um banco de dados, serão geradas entidades, relacionamentos, atributos e seus respectivos tipos de dados.

Como o objetivo desta disciplina é aprender conceitos de banco de dados, utilizaremos a **ferramenta CASE conhecida como EERCASE (Entidade-Relacionamento Estendido CASE)** (ALVES et al. 2014), que pode ser baixada pelo endereço: https://sites.google.com/a/cin.ufpe.br/eercase/download

Na Figura 20 é apresentada a interface gráfica da ferramenta EERCASE. Nesta figura é possível verificar que: 1) na área especificada pela letra “A” têm-se os artefatos produzidos pelo projetista; 2) na área especificada pela letra “B” tem-se o espaço de desenvolvimento dos esquemas; 3) na área especificada pela letra “C” têm-se as propriedades do componente EER selecionado, podendo o projetista modificá-las de acordo com as necessidades; 4) por fim, na área especificada pela letra “D” têm-se os construtores do EER.

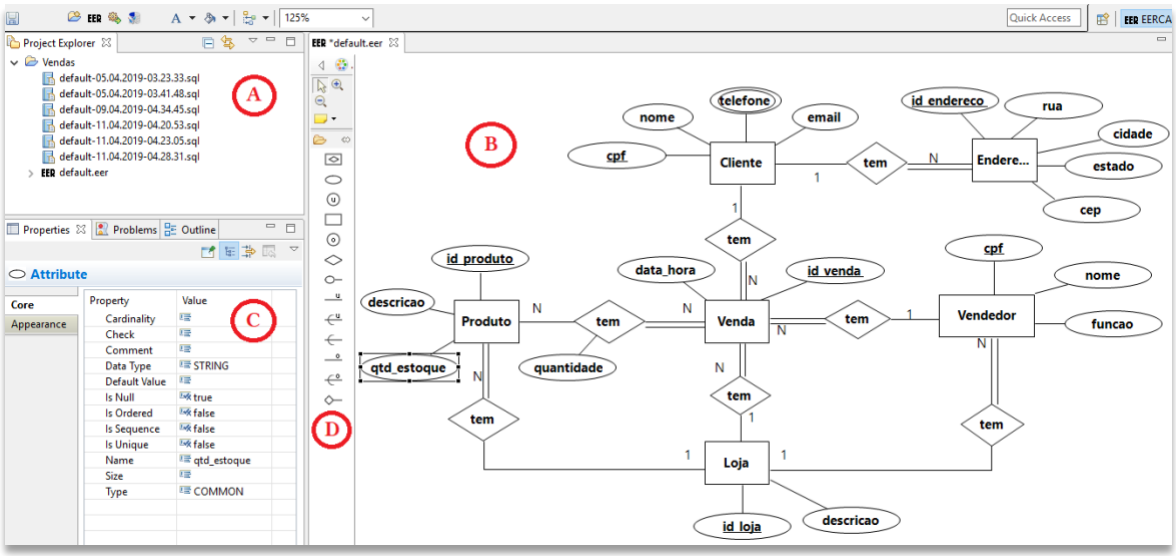

Figura 20 - Ferramenta EERCASE.
Fonte: Alves et al. (2014)
Descrição: Nesta figura verificam-se as principais áreas da ferramenta EERCASE, sendo o ponto 'A' a área que contêm os artefatos produzidos; o ponto 'B' a área de desenvolvimento; o ponto 'C' as propriedades do item selecionado; e o ponto 'D' os construtores do modelo EER para utilização.

Para você criar um esquema na ferramenta EERCASE, você deve seguir os seguintes passos:

1. Baixe a última versão do Java no site: https://www.java.com/pt_BR/download/;
2. Baixe a ferramenta EERCASE no site: https://sites.google.com/a/cin.ufpe.br/eercase/download;
3. Ao descompactar o arquivo ".rar" do EERCASE, abra a ferramenta dando um duplo clique no arquivo eercase.exe;
4. Ao abrir a ferramenta, crie um novo projeto clicando no ícone no canto superior esquerdo; Na área "Project Explorer", selecione o projeto que você acabou de criar e clique no ícone EER no canto superior esquerdo; Quando aparecer a área de modelagem de esquemas (área "B" da Figura 20), você deve escolher um construtor EER na área "D" (Figura 20) e clicar na área de desenvolvimento, área "B";
5. Para inserir uma entidade, clique no construtor "Entity" na área "D" e clique sobre a área de desenvolvimento, área "B"; insira um nome para esta entidade.
6. Para inserir um atributo na entidade que você criou no passo anterior, clique no construtor "Attribute" que fica na área "D" e clique na área de desenvolvimento "B",

próximo à entidade que você criou, e insira um nome. Clique no construtor “Attribute Link” na área “D”, clique (e segure o clique) sobre o atributo e arraste o mouse até a entidade que você criou no passo anterior. Ao final você deve ter uma entidade com um atributo. Para inserir o tipo do atributo, selecione o atributo, e na área “C” da Figura 20, você deve escolher o tipo alterando o campo “Data Type”;

7. Repita os passos 7 e 8 para outras entidades;
8. Para criar um relacionamento entre entidades, selecione o construtor “Relationship” na área “D” e clique na área de desenvolvimento, área “B”. Clique no construtor “Relationship Link” na área “D” e faça uma ligação entre o relacionamento e a entidade, clicando em um e soltando o clique no outro (clique e segure). Você pode clicar e segurar primeiro na entidade e soltar no relacionamento; ou iniciar pelo relacionamento. Altere a cardinalidade do relacionamento clicando no seu link, e na área de propriedades modifique o campo “Cardinality”.

Como atividade para exercitar a ferramenta EERCASE, crie o esquema visto na Figura 17. Uma dica: verifique sempre as propriedades do construtor na área “C” da Figura 20. Bons modelos!

2.5 Ferramenta CASE – StarUML

Como já foi dito, a linguagem UML é também usada para modelar bancos de dados. Como sugestão, utilizaremos a ferramenta StarUML, que pode ser adquirida pelo link: http://staruml.io/download. Você pode utilizar uma versão não registrada, mas a ferramenta informa que “para o uso continuado, você deve adquirir uma licença e registrar o software”. Na figura abaixo é apresentada a interface gráfica da ferramenta StarUML e suas respectivas áreas de navegação e desenvolvimento dos esquemas.

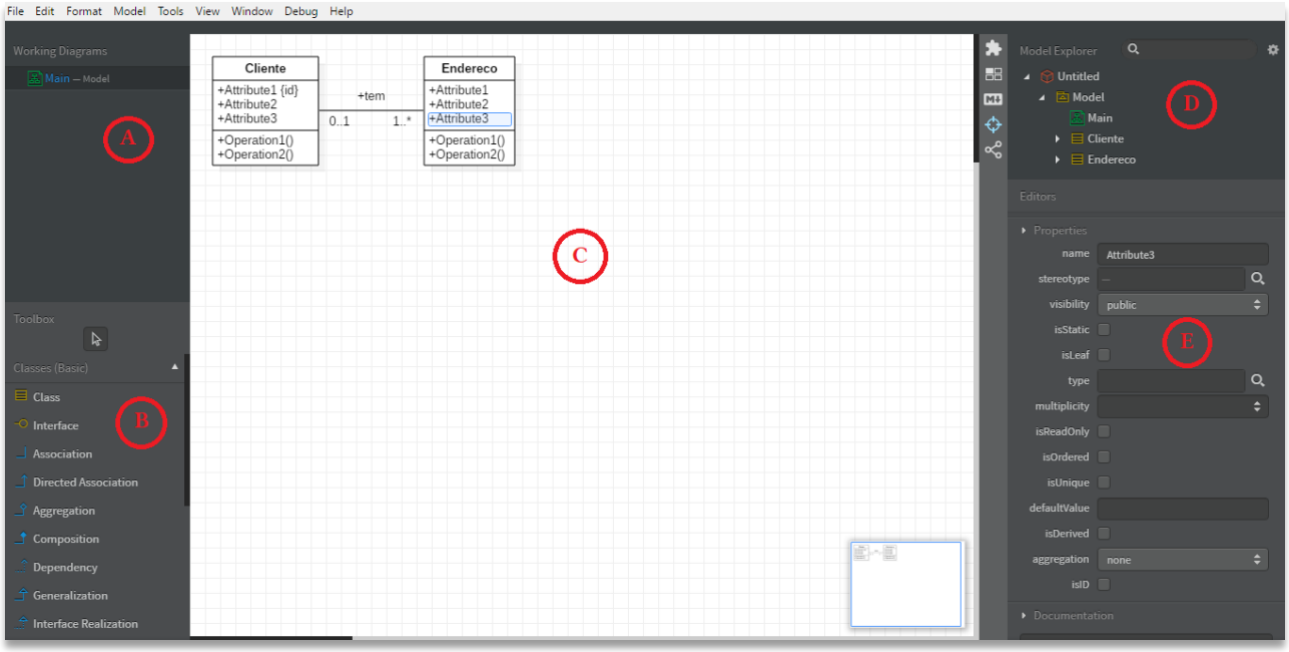

Figura 21 - Ferramenta StarUML
Fonte: O Autor
Descrição: Nesta figura é possível verificar as principais áreas de ferramenta StarUML, sendo o ponto 'A' a área que contêm os artefatos produzidos; o ponto 'B' os construtores da linguagem UML (Diagrama de Classe); o ponto 'C' a área de desenvolvimento; a área 'D' exibição dos items do modelo em desenvolvimento; e o ponto 'E' as propriedades de cada item.

Na Figura 21 é possível identificar as cinco áreas da ferramenta StarUML: 1) na área identificada pela letra "A", o projetista tem acesso aos esquemas produzidos; 2) na área identificada pela letra "B", ficam disponibilizados os construtores do Diagrama de Classe UML; 3) a área identificada pela letra "C" é a área de trabalho, onde o projetista desenvolve efetivamente um esquema de sistema; 4) a área identificada pela letra "D" é a área onde o projetista pode navegar entre os componentes que estão modelados na área de trabalho; 5) na área identificada pela letra "E", é possível alterar as propriedades de cada componente que está selecionado na área de trabalho.

Para você criar um esquema de banco de dados com a ferramenta StarUML, siga os seguintes passos:

1. Baixe a ferramenta no link: http://staruml.io/download e instale;
2. Ao abrir a ferramenta, você já vai ter acesso ao ambiente de desenvolvimento;
3. Para criar uma classe, navegue para área "B" da Figura 21, escolha o construtor "Class", e insira-o na área de desenvolvimento, área "C"; insira um nome para a nova classe;

4. Para inserir atributos nesta classe recém criada, clique com o lado direito do mouse sobre a classe e escolha “Add/Atribute”; para inserir um nome nele, selecione-o e navegue até a área “E” de propriedades; modifique o nome no campo “Name” e modifique também o tipo no campo “Type”;
5. Crie uma outra classe seguindo os passos 3 e 4;
6. Para criar um relacionamento, navegue até a área “B” e escolha o construtor “Association” (a UML chama um relacionamento de Associação); faça uma ligação entre as duas classes clicando sobre uma e soltando na outra; insira um nome para este relacionamento.

Como atividade para exercitar a ferramenta StarUML, reproduza em UML o esquema visto na Figura 17, que está em ER. Bom trabalho!

Existem várias **ferramentas para modelar bancos de dados utilizando UML**. Sugiro que você faça uma pesquisa no google e procure uma que melhor atenda às suas necessidades

2.6 Transformação de Modelo Conceitual para SQL

Até aqui imagino que você já criou um artefato a partir das ferramentas do tipo CASE vistas na seção anterior. Após a criação destes artefatos, é preciso então gerar os comandos SQL a partir destes modelos. Por exemplo, uma entidade modelada pela ferramenta EERCASE deve representar um comando SQL-DDL que define uma tabela em um banco de dados relacional. Sendo assim, um script “CREATE TABLE CLIENTE...” deve ser gerado e executado em um SGBD.

Tomando como exemplo o banco de dados Vendas, cujo esquema foi mostrado na Figura 17, iremos gerar seus respectivos comandos SQL-DDL. Primeiramente explicaremos os relacionamentos: 1xN e NxN. Na Figura 22 abaixo, é possível verificar uma parte do esquema do BD Vendas, onde há um relacionamento de Cardinalidade 1xN entre as entidades Cliente e Endereco. Já a Figura 24 também se trata de uma parte do esquema do BD Vendas e foca o relacionamento entre as entidades Produto e Venda.

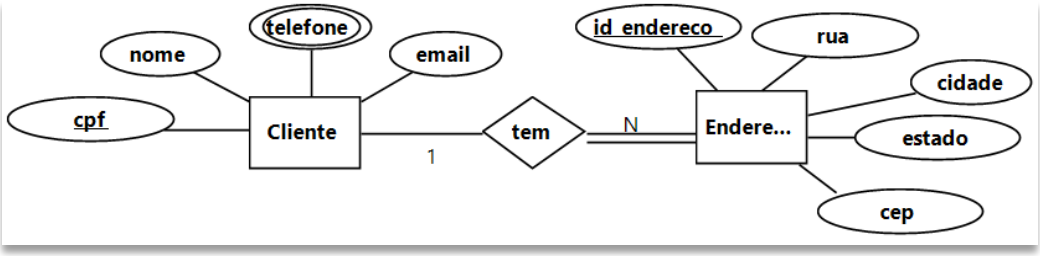

Figura 22 - Relacionamento Binário entre as entidades Cliente e Endereço do banco de dados Vendas.
Fonte: O Autor.
Descrição: Relacionamento Binário entre as entidades Cliente e Endereço, cuja cardinalidade é de 1xN. A entidade Cliente contém os atributos: CPF (chave primária), Nome, Telefone e Email. Já a entidade Endereço contém os atributos: Id_endereco (chave primária), Rua, Cidade, Estado e Cep.

No esquema mostrado na Figura 22, é possível verificar algumas particularidades do modelo ER, a saber o atributo multivalorado "Telefone" e o relacionamento com participação total por parte da entidade Endereco. No primeiro caso, no tocante ao atributo multivalorado, um cliente pode ter mais de um telefone. Para se transformar isso para SQL, não é interessante criar colunas de forma que limite a quantidade de telefones de um cliente, uma vez que você não sabe quantos telefones o cliente vai ter (pode ser 0, 1, 2, N; telefones). Sendo assim, atributos multivalorados são transformados como uma tabela em banco de dados, e a chave primária da tabela Cliente se torna chave estrangeira na tabela telefone. Com isso, você pode dizer que, não importa a quantidade de telefones que o cliente pode ter, ele vai poder cadastrar todos, ou até mesmo nenhum. Por outro lado, se colocássemos colunas nas tabelas, teríamos: Telefone1, Telefone2, Telefone3; neste caso, se um cliente não tivesse nenhum telefone, estes atributos ficariam nulos, sem nenhuma informação. A transformação para SQL da entidade Cliente pode ser vista na Figura 6, da Competência 1. Já o relacionamento "tem" de 1xN entre Cliente e Endereco, nos diz que um Cliente pode ter 1 ou mais endereços. Esse "pode ter" é devido à participação da entidade Cliente não ser obrigatória na relação, ou seja, ela não é TOTAL, como é o caso do Endereço. Um cliente, para ser cadastrado, não precisa necessariamente de um endereço. Contudo, um endereço, para ser cadastrado, precisa necessariamente de um cliente. Transformando para SQL, a chave primária da tabela Cliente se torna chave estrangeira na tabela Endereco. A seguir, Na Figura 23, é possível verificar a transformação da entidade Endereco para SQL.

```
CREATE TABLE ENDERECO (
    PKENDERECO INTEGER PRIMARY KEY,
    CPF VARCHAR(15),
    RUA_NUMERO VARCHAR(50) NOT NULL,
    BAIRRO VARCHAR(30) NOT NULL,
    MUNICIPIO VARCHAR(30) NOT NULL,
    ESTADO CHAR(2),
    CEP VARCHAR(10),
    FOREIGN KEY (CPF) REFERENCES CLIENTE (PKCPF)
    );
```

Figura 23: Script SQL-DDL da tabela Endereço.
Fonte: O Autor.
Descrição: Nesta figura contém o script de criação da tabela Endereco do banco de dados Vendas. Ela contém os atributos: PKEndereco (chave primária), CPF (chave estrangeira), Rua_Numero, Bairro, Município, Estado e Cep.

Observe que a Figura 23 representa exatamente a estrutura modelada pela ferramenta do tipo CASE EERCASE. Gostaria que você observasse os seguintes pontos nesta figura:

- A coluna CPF deve ser do mesmo tipo da chave primária na qual ela está se referenciando, que neste caso, se trata da chave primária PKCPF da tabela Cliente;
- A coluna CPF é uma chave estrangeira (observe as palavras FOREIGN KEY REFERENCES);
- A coluna RUA_NUMERO está em um mesmo campo. No entanto, o usuário pode criar um atributo para cada campo. Caso fossem criados dois outros atributos Rua e Número, salientando que o atributo Número, para este caso, não seria necessário tê-lo como um valor do tipo Inteiro, uma vez que o número da casa não é usado para realizar nenhum cálculo. Portanto, como regra, caso o atributo não seja chave primária, só insira valores numéricos caso haja algum cálculo com este campo;
- Os atributos Bairro, Município e Estado, embora sejam aqui colunas, poderiam ser representados por tabelas. Você vai perceber que, ao inserir algum dado nesta tabela, haverá repetições e, consequentemente, redundância de dados. Por exemplo, se inserirmos todos os habitantes do bairro “Boa Vista”, a tabela conterá o termo “Boa Vista” em cada linha da tabela. Por outro lado, poderia criar uma outra tabela chamada “Bairros”, cuja chave primária representaria esta entidade ao ser inserida na tabela Endereco como chave estrangeira (FK_BAIRRO, por exemplo), substituindo o termo literal do nome do bairro como atributo na tabela Endereco. O

mesmo pode ser feito para os atributos Estado e Município. Estas características são abordadas mais à frente e são conhecidas como Dependências Funcionais.

Na próxima figura, abordaremos um caso mais crítico de transformação, onde duas tabelas possuem cardinalidade NxN e, também, um atributo inserido no relacionamento.

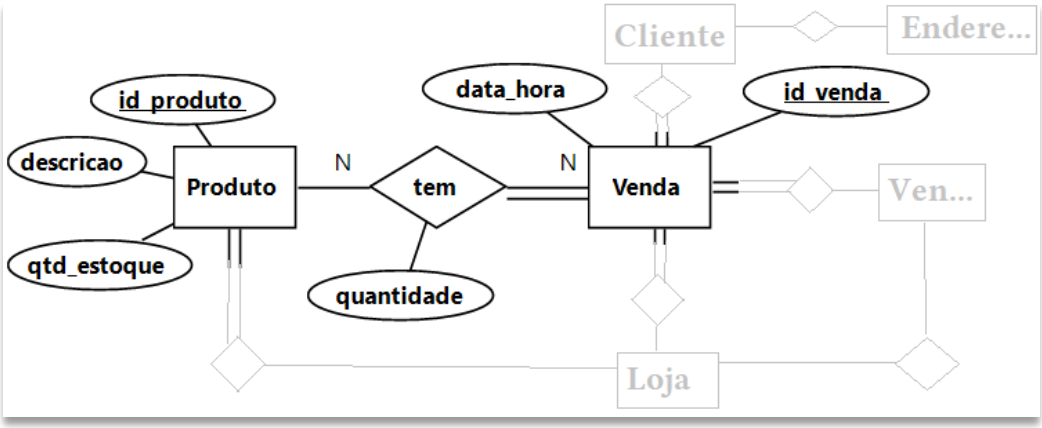

Figura 24 - Relacionamento Binário de Cardinalidade NxN entre as entidades Produto e Venda, do Banco de Dados Vendas.
Fonte: O Autor.
Descrição: Nesta figura contém um relacionamento binário, NxN, entre as entidades Produto e Venda. A entidade Produto contém os atributos: Id_produto (chave primária), Descrição e Qtd_Estoque. Já a tabela Venda contém os atributos Id_venda e Data_Hora. O relacionamento contém um atributo chamado Quantidade.

A Figura 24 representa o esquema do banco de dados Vendas visto na Competência 1, sendo o relacionamento entre a entidade Produto e Venda destacado. Nesta figura, observa-se o relacionamento de cardinalidade NxN, ou seja, muitos para muitos, onde neste caso, **um produto pode estar em várias vendas e uma venda pode ter vários produtos**. Para transformar esta modelagem é preciso saber que só temos a estrutura de tabela formada por linhas e colunas. Com isso, caso transformássemos igual ao relacionamento de cardinalidade de 1xN, teríamos a chave primária do Produto na tabela Venda como chave estrangeira. No entanto, se isso fosse feito, a tabela Venda ficaria sem chave primária, pois a cada produto inserido teríamos uma nova linha e a unicidade da chave primária não seria respeitada. Veja no exemplo da Figura 25 como seria se transformássemos a tabela Venda igual a um relacionamento 1xN com a tabela Produto.

Venda

ID_Venda	Data_Hora	FK_Produto
1	11/10/2018 10:42:00	535
1	11/10/2018 10:42:00	839
1	11/10/2018 10:42:00	320
1	11/10/2018 10:42:00	256
2	04/12/2018 15:36:22	658
2	04/12/2018 15:36:22	495
2	04/12/2018 15:36:22	687

Produto

ID_Produto	Descricao	Valor_Unitario	QTD_Estoque
535	Produto A	2,50	2021
839	Produto B	33,59	5326
320	Produto C	54,25	1223
256	Produto D	200,00	2456

Figura 25 - Erro de redundância de dados visto em relacionamento NxN modelado como um 1xN.
Fonte: O Autor.
Descrição: Esta figura contém duas tabelas: Venda e Produto; sendo a tabela Venda modelada erradamente igual a um relacionamento 1xN, resultando no erro de redundância de dados. A tabela produto contém alguns registros fictícios com relacionamento na tabela Venda.

A Figura 25 mostra o que acontece se for transformar um relacionamento de cardinalidade NxN como se fosse um relacionamento de 1xN. Observe que o identificador da Venda se repete, perdendo a sua função principal que é de manter um identificador único para cada Venda. Isso implicaria em redundância de dados e falta de integridade referencial. Em casos onde há um relacionamento de cardinalidade NxN, é necessário criar uma **terceira tabela**, comportando as chaves primárias de ambas. Conforme mostrado na Figura 5 da Competência 1, as tabelas Produtos e Venda possuem um registro para cada instância de entidade. Já a tabela identificada como Venda_Produto (Figura 5), contém os atributos FKVENDA, FKPRODUTO e Quantidade; que são respectivamente: a chave estrangeira da venda, chave estrangeira do produto e a quantidade do produto em questão. A chave primária da tabela Venda_Produto será uma chave composta (duas ou mais chaves) que contém duas super-chaves: FKVENDAS e FKPRODUTO. A combinação destas duas chaves formam uma chave única jamais repetida na tabela. Uma super-chave de uma Tabela pode ser definida como um conjunto de colunas (atributos) da tabela com a propriedade de identificação unívoca de linhas da tabela.Em outras palavras não podem existir duas ou mais linhas da tabela com o(s) mesmo(s)

valores de uma super-chave.. O objetivo de se criar uma terceira tabela é garantir que os dados serão inseridos sem redundâncias e com integridade referencial. Na Figura 26 é possível verificar três scripts SQL-DDL das tabelas Produto, Venda e Venda_Produto.

```
CREATE TABLE PRODUTO (
    PKPRODUTO INTEGER PRIMARY KEY,
    DESCRICAO VARCHAR(50) NOT NULL,
    QTD_ESTOQUE INTEGER NOT NULL,
    UNIDADE_ARMAZENADA VARCHAR(20),
    FKLOJA INTEGER,
    VALOR_UNIDADE DECIMAL(10,2),
    FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA)
    );
```

A

```
CREATE TABLE VENDA (
    PKVENDA INTEGER PRIMARY KEY,
    DATA_HORA TIMESTAMP,
    VALOR DECIMAL(10,2),
    FKCLIENTE VARCHAR(15),
    FKLOJA INTEGER,
    FKVENDEDOR VARCHAR(15),
    FOREIGN KEY (FKCLIENTE) REFERENCES CLIENTE (PKCPF),
    FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA),
    FOREIGN KEY (FKVENDEDOR) REFERENCES VENDEDOR (PKCPF)
    );
```

B

```
CREATE TABLE VENDA_PRODUTO (
    FKVENDA INTEGER,
    FKPRODUTO INTEGER,
    QUANTIDADE INTEGER,
    PRIMARY KEY (FKVENDA, FKPRODUTO),
    FOREIGN KEY (FKVENDA) REFERENCES VENDA (PKVENDA),
    FOREIGN KEY (FKPRODUTO) REFERENCES PRODUTO (PKPRODUTO)
    );
```

C

Figura 26 - SQL-DDL do Relacionamento NxN entre as entidades Venda e Produto.
Fonte: O Autor.
Descrição: A tabela Produto 'A' contém os atributos: PKProduto (chave primária), Descrição, Qtd_Estoque, Unidade_armazenada, FK_Loja (chave estrangeira), e Valor_Unidade.

A Figura 26 representa a transformação para SQL-DDL da modelagem vista na Figura 24 (relacionamento NxN). Observe que a Figura 26 contém três scripts do tipo CREATE TABLE: A, B e C. O script "A" cria a tabela Produto. No entanto, caso você execute este script no SGBD Postgres, não vai funcionar. Se você observar atentamente vai perceber que a tabela Produto contém uma chave estrangeira FKLOJA, que é chave primária da tabela Loja. É importante você atentar para isso: antes

de criar uma tabela, verifique se esta contém uma chave estrangeira de outra tabela. Se for este o caso, você precisa executar o script que cria esta última tabela (Loja) para, assim, executar a criação da anterior (Produto). Portanto, execute primeiramente o script da tabela Loja (Figura 27, abaixo) e em seguida o script da tabela Produto (script "A" da Figura 26). O mesmo vale para o script "B" da Figura 26, que cria a tabela Venda, cujas chaves estrangeiras são referentes às tabelas: Cliente, Loja e Vendedor. Por último, tem-se a criação da tabela Venda_Produto, que contém, basicamente, o atributo QUANTIDADE e as chaves estrangeiras FKVENDA e FKPRODUTO compondo uma chave primária.

```
CREATE TABLE LOJA (
    PKLOJA INTEGER PRIMARY KEY,
    DESCRICAO VARCHAR(50)
    );
```

Figura 27 - Script SQL-DDL da tabela Loja.
Fonte: O Autor.
Descrição: Esta figura contém um script SQL-DDL da tabela Loja, com os atributos: PK_loja (chave primária) e Descrição.

Aqui temos dois conceitos: **Chave Composta e Chave Estrangeira-Primária**. Como já dito, chave composta é uma chave identificadora que contém mais de uma chave, neste caso são duas chaves que compõem a chave primária. E, o conceito de chave estrangeira-primária, que é uma chave criada originalmente como primária em uma primeira tabela e se torna estrangeira em outra, cuja condição também é de chave primária. Podemos dar como exemplo quando temos a chave primária da tabela Pessoa, PKCPF, e queremos criar uma tabela para armazenar os telefones desta pessoa. Assim, podemos dizer que a chave primária PKCPF será a chave estrangeira em Telefone (FKCPF), junto ao número de telefone, formando uma chave composta sendo uma delas estrangeira (FKCPF).

Por fim, como um último script de transformação da modelagem vista para o banco de dados Vendas (representado na Figura 17), tem-se o script SQL-DDL para criar a tabela VENDEDOR, conforme a Figura 28 contida abaixo:

```
CREATE TABLE VENDEDOR (
    PKCPF VARCHAR(15) PRIMARY KEY,
    NOME VARCHAR(50),
    FKLOJA INTEGER,
    FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA)
    );
```

Figura 28 - Script SQL-DDL da tabela Vendedor.
Fonte: O Autor.
Descrição: Nesta figura contém um Script SQL-DDL da tabela Vendedor, com os atributos: PKCpf (chave primária), Nome e FKLoja (chave estrangeira).

É indispensável que você crie estas tabelas em um SGBD, isso vai contribuir para um melhor domínio sobre projetos de bancos de dados e manipulação de dados.

2.7 Dependência Funcional

Dependência Funcional é um conceito de banco de dados que está relacionado a modelos relacionais de dados eficientes, de forma que, aplicando regras conhecidas como normalização de tabelas, tenta evitar os seguintes problemas: semântica do esquema de dados, reduzir redundâncias, reduzir valores nulos, evitar valores falsos para junções de tabelas, e evitar anomalias de atualizações.

Diz-se que um atributo qualquer “A” depende funcionalmente de outro atributo “B” quando este último consegue representá-lo no mundo real. Por exemplo, na tabela Produto (figura 5), do banco de dados Vendas, os atributos Descrição do produto, Quantidade em estoque, Unidade Armazenada, e Valor da Unidade; são atributos que dependem funcionalmente do número de série do produto (PKPRODUTO). Caso fosse inserido algum dos seguintes atributos: Descrição do Fornecedor, Estado Origem, Marca, ou outro que não tenha ligação direta com o número de série do produto (PKPRODUTO), ele não dependeria funcionalmente de sua chave primária PKPRODUTO. Sendo assim, de forma objetiva, o projetista de banco de dados deve separar as “sub-tabelas” que estão dentro desta tabela, com uma operação chamada normalização.

Neste exemplo citado, três outras tabelas seriam formadas: Fornecedor, Estado e Marca. A Figura 29 mostra este exemplo com a tabela Produto **desnormalizada,** e a Figura 30 mostra o mesmo exemplo com o processo de **normalização** realizado.

Produto

PKPRODUTO	Descricao	Valor_Unitario	Unidade_Medida	Qtd_Estoque	Peso (gr)	Fornecedor	Estado_Origem	Marca
5468	Feijão Du Caldo	10,00	KG	138	1000	RL Distribuidora	Pernambuco	Tio José
5469	Papel Higiênico	2,00	Pacote C/ 4	253	20	Minas Alimentos	Minas Gerais	Levis
5470	Sabão Barra	5,00	Pacote C/5	126	30	Minas Alimentos	Minas Gerais	Bem Te Visto
5471	Jogo de Panela	250,00	Caixa C/6	52	1200	Salvador Produtos	Bahia	Tramontim

Figura 29 - Exemplo de tabela desnormalizada.
Fonte: O Autor.
Descrição: Tabela produto desnormalizada com os atributos: PKProduto, Descrição, Valor_Unitario, Unidade_Medida, WTD_Estoque, Peso (gr), Fornecedor, Estado_Origem e Marca.

Produto

PKPRODUTO	Descricao	Valor_Unitario	Unidade_Medida	Qtd_Estoque	Peso (gr)	FKFORNECEDOR	FKESTADO	FKMARCA
5468	Feijão Du Caldo	10,00	KG	138	1000	2345	12	5234
5469	Papel Higiênico	2,00	Pacote C/ 4	253	20	4938	13	6896
5470	Sabão Barra	5,00	Pacote C/5	126	30	4938	13	2546
5471	Jogo de Panela	250,00	Caixa C/6	52	1200	9084	14	3728

Fornecedor

PKFORNECEDOR	Razao_Social	Outros
2345	RL Distribuidora	...
4938	Minas Alimentos	...
9084	Salvador Produtos	..

Estado

PKESTADO	Descricao	UF	Outros
12	Pernambuco	PE	...
13	Minas Gerais	MG	...
14	Bahia	BA	...

Marca

PKMARCA	Descricao	Outros
5234	Tio José	...
6896	Levis	...
2546	Bem Te Visto	...
3728	Tramontim	...

Figura 30 - Processo de normalização.
Fonte: O Autor.
Descrição: Contém o processo de normalização, partindo da tabela Produto desnormalizada, sendo geradas as tabelas: Fornecedor, Estado e Marca.

A Figura 29 mostra uma tabela Produto contendo atributos diversos. Os atributos: Descrição, Valor_Unitario, Unidade_Medida, QTD_Estoque, e Peso (gr); de fato, dependem funcionalmente de sua chave primária PKPRODUTO, levando em consideração que esta chave representa um identificador único de um produto. Já os atributos: Fornecedor, Estado_Origem e Marca, embora tenham alguma relação com um produto, não possuem uma dependência funcional

com este, uma vez que podem, facilmente, ser representados por uma entidade separada. Neste sentido, deve-se normalizar a tabela Produto, de forma que os atributos: Fornecedor, Estado_origem e Marca serão transformados em uma entidade cada, e suas respectivas chaves primárias formarão chaves estrangeiras para a tabela de Produto (Figura 30).

É importante destacar que há uma coerência nos nomes dos atributos. Ou seja, a descrição do atributo informa de fato o que ele representa, o que facilita o processo de normalização. Não faria sentido normalizar colunas que não são autoexplicativas, por exemplo: um código, sigla, etc. Em geral, quanto mais fácil for a interpretação dos atributos e nomes de tabelas, melhor será a normalização.

2.8 Formas Normais

Ao realizar o desmembramento de entidades contidas em tabelas, o projetista de banco de dados não sabe, a rigor, se tal processo está ou não de acordo com os padrões exigidos para um bom projeto de banco de dados. Estou querendo dizer que, muito embora você percebeu que havia entidades dentro de uma mesma tabela, não significa que estas tabelas estão de fato eficientes. O que fazer, então? É necessário realizar a normalização, que é o processo de modelar o banco de dados, usando um conjunto de regras, chamados de formas normais, e cujo o propósito é reestruturar tabelas e atributos, eliminando, ou pelo menos reduzindo as redundâncias.

Existe na literatura o conceito de Forma Normal, que é dividido em: primeira forma normal (1FN), segunda forma normal (2FN), terceira forma normal (3FN), forma normal de Boyce-Codd (FNBC ou BCNF), quarta forma normal (4FN) e quinta forma normal (5FN). Embora existam estas cinco formas normais, um projeto de banco de dados está em um nível aceitável de qualidade de normalização se estiver no mínimo na 3FN, que serão abordadas neste curso.

2.8.1 Primeira Forma Normal

A primeira forma normal (1FN) diz que toda tabela deve conter atributos **atômicos**, isto é, não deve ter atributos multivalorados e atributos compostos em um mesmo campo. Um exemplo de atributo composto é uma coluna contendo o DDD (Discagem Direta a Distância) + Número de Telefone, distintos e no mesmo campo: (081; 993029882), ou multivalorado (81-993827788; 81-

989887799; 81-998890099). Como solução, é preciso remover este tipo de atributo, mantendo, neste caso, os números de telefones na mesma tabela (não se trata ainda da melhor solução). Observe que a tabela da Figura 31 A fere a regra da 1FN, e uma possível solução é mostrada na Figura 31B.

PK_Cliente_CPF	Nome	Telefone
99222888909	José Martins	81 – 949399xx; 82 – 980033xx; 81 – 9988977xx;
90808229988	Maria da Silva	21 – 320944xx; 11 – 789922xx;
00022930299	Cláudio José	41 – 990933xx; 42 – 990033xx;
90009922288	Sérgio Santos	31 – 879933xx;

A

PK_Cliente_CPF	Nome	Telefone
99222888909	José Martins	81949399xx
99222888909	José Martins	82980033xx
99222888909	José Martins	819988977xx
90808229988	Maria da Silva	21320944xx
90808229988	Maria da Silva	11789922xx
00022930299	Cláudio José	42990033xx
00022930299	Cláudio José	41990933xx
90009922288	Sérgio Santos	31879933xx

B

Figura 31 - Primeira Forma Normal
Fonte: O Autor.
Descrição: Esta figura contém duas tabelas, sendo a primeira 'A' uma tabela que fere a primeira forma normal, e a segunda 'B' sem o problema de dados multivalorados.

Na Figura 31 é possível verificar duas tabelas, sendo a Figura 31A um exemplo de tabela que fere a regra da 1FN e a Figura 31B uma possível solução. Observe que a Figura 31B possui uma coluna chamada Telefone, que contém todos os registros que antes estavam em um único campo. Com essa transformação, a chave primária da tabela, que antes era apenas PK_CLIENTE_CPF, agora é uma chave composta: PK_Cliente_CPF + Telefone (verifique o sublinhado); que neste caso, formam um número único com integridade de chave. Contudo, muito embora a primeira forma normal já esteja solucionada e respeitada, ainda assim, a tabela resultante (Figura 31B) contém redundância de dados e fere a segunda forma normal, que veremos a seguir.

2.8.2 Segunda Forma Normal

A segunda forma normal diz que, para que uma tabela esteja na segunda forma normal (2FN), é preciso que esta esteja na primeira forma normal, e também que: **os registros na tabela não**

devem depender de algo que não seja a chave primária da referida tabela. Ou seja, no exemplo mostrado na Figura 31 anterior, o atributo Telefone da tabela da Figura 31B, fere esta regra, uma vez que o telefone não é um atributo inerente de um cliente (ser humano). Os atributos que podem ser de uma pessoa/cliente, por exemplo são: Cor, Altura, Sexo, Estado Civil, etc. O atributo Telefone, embora esteja vinculado a um cliente, fere a 2FN. Como solução, deve-se criar uma tabela contendo os telefones (DDD+Telefone juntos, ex: 819978933XX) dos seus respectivos donos, cuja chave primária se torna uma chave composta (CPF+Telefone), onde um cliente pode ter um ou mais telefones sem que haja problemas de recuperação e atualização de dados. Para melhor ilustrar este cenário, vamos pegar a tabela resultante da 1FN (Figura 31B) e vamos subdividi-las, formando outras duas tabelas, que podem ser vistas a seguir na Figura 32.

Figura 32 - Segunda Forma Normal.
Fonte: O Autor.
Descrição: Esta figura contém três tabelas, sendo a primeira 'A' uma tabela que fere a segunda forma normal, retirado o problema criando duas outras, sendo a 'B' contendo dados do cliente e a segunda dados do telefone.

Na Figura 32 é possível verificar três tabelas, sendo a tabela "A" a mesma que foi derivada da primeira forma normal vista anteriormente, e as tabelas "B" e "C", que representam a solução para a segunda forma normal. Observe que antes da transformação (tabela A) os dados se

repetem, tornando um banco de dados redundante e ineficiente em casos de recuperação e atualização de informações. Por exemplo, caso fosse necessário atualizar o nome do cliente na tabela “A”, a operação de UPDATE alteraria três registros, desnecessariamente. Já na tabela “B”, os dados do cliente ficam independentes, tornando operações mais eficientes, claras e organizadas. É importante você saber criar modelos de dados eficientes, isso evita problemas de desempenho, redundância e outros já mencionados anteriormente.

2.8.3 Terceira Forma Normal

Este conceito requer que você preste bastante atenção. A terceira forma normal está baseada em dependência transitiva. O conceito de transitividade está relacionado ao atributo não chave que é dependente funcionalmente de outro atributo que não é chave principal, mas de alguma forma está vinculada à chave principal. Para ficar mais claro, observe que a tabela na Figura 33 abaixo possui os atributos: PK_Cliente_CPF, Nome, Data_Nascimento, ID_Preferência_Compra, Preferência_Cor e Preferência_Tamanho. Destes atributos, a chave primária é o CPF, e os atributos Nome e Data_Nascimento dependem funcionalmente desta chave primária. Contudo, os atributos Preferencia_Cor e Preferencia_Tamanho, embora estejam vinculados à preferência do cliente, não dependem funcionalmente do seu CPF, mas sim do atributo ID_Preferencia_Compra.

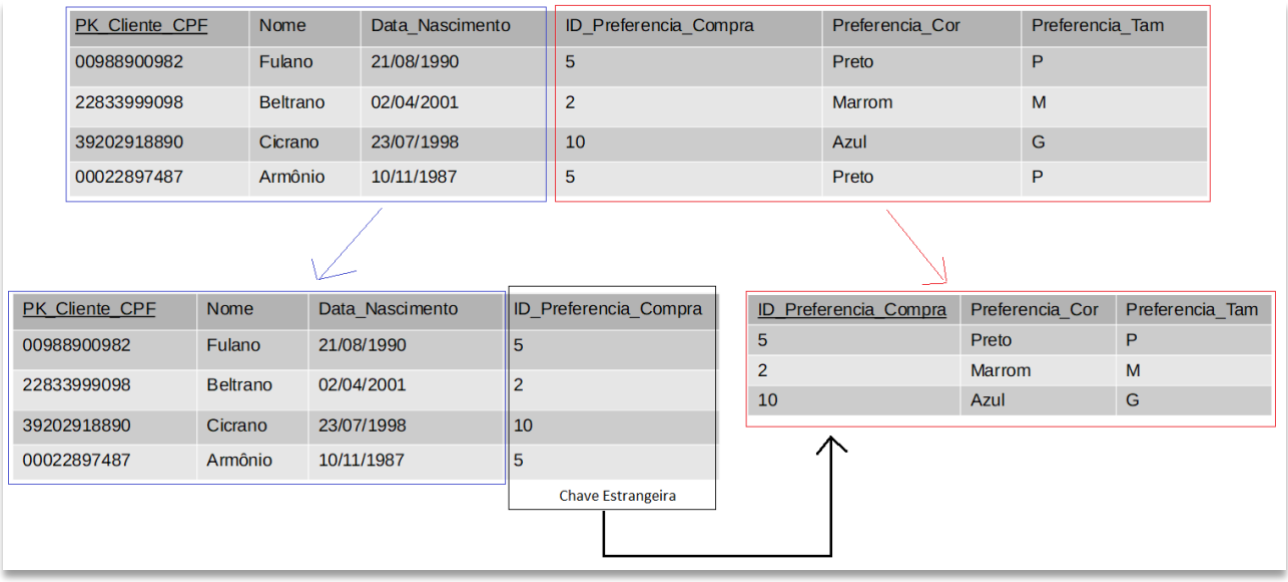

Figura 33 - Terceira Forma Normal.

Fonte: O Autor.
Descrição: Esta figura contém três tabelas, sendo a primeira uma tabela que fere a terceira forma normal, sendo decomposta em duas outras, uma contendo os dados do cliente, e a outra com dados de preferências de compra.

Para aplicar a terceira forma normal deve-se retirar a dependência transitiva, criando duas tabelas para cada entidade, referenciando suas respectivas chaves (Figura 33).

A seguir, na Figura 34 é possível verificar um resumo para aplicação das formas normais vistas anteriormente.

Formas Normais	Teste a ser feito	Ação de Normalização
Primeira Forma Normal	A tabela não deve conter atributos não atômicos ou tabelas aninhadas, ou seja, deve conter apenas atributos atômicos.	Retirar a não atomicidade do dado, por outra tabela ou por reestruturação.
Segunda Forma Normal	Para tabelas que possuem chaves primárias com vários atributos, estes atributos devem ter dependência funcional total a chave primária.	Deve decompor e montar uma nova tabela para cada chave junto com seus atributos.
Terceira Forma Normal	Não deve ter dependência transitiva entre um atributo não-chave e uma chave primária.	Deve decompor e montar uma tabela que contenha os atributos que formam a transitividade, e vincular esta tabela à primeira na qual foi derivada.

Tabela 1 - Formas Normais, Teste a ser feito e Ação de Normalização.
Fonte: Elmasri e Navathe (2005), adaptado.
Descrição: Esta tabela contém um resumo das formas normais, com o teste a ser feito e a ação para resolução do respectivo problema.

3.Competência 03 | Construir Tabelas e Dicionário de Dados de um Banco de Dados

Até aqui você criou uma estrutura de dados a partir de um minimundo e definiu scripts SQL-DDL para persistir os dados em um SGBD. Nesta competência você irá efetivamente criar um banco de dados (dicionário de dados disponível para operações) e manipular dados fictícios, simulando transações de sistemas diversos.

Para criação de **dicionário de dados**
veja em **material complementar**

3.1 Instalação de um SGBD e criação de um Banco de Dados

Conforme informamos na primeira competência, você deve instalar um SGBD em seu computador. Você pode instalar o SQLite ou o PostgreSQL versão 9.6.14.

3.1.1 Baixar e Operacionalizar o SQLite

O SQLite é um SGBD leve que gerencia dados diretamente em arquivos, tornando simples a leitura, gravação e manipulação desse conteúdo. O SQLite é indicado para projetos não críticos, como por exemplo: App mobiles, internet das coisas, aplicações com arquivos, pequenos web sites, ou para testes. Este banco de dados não requer instalação, basta você baixar e utilizar. Para isso siga os seguintes passos:

1. Baixe o arquivo a seguir e descompacte-o em uma pasta:
    - https://www.sqlite.org/2019/sqlite-tools-win32-x86-3280000.zip
2. Os arquivos baixados são:
    - sqldiff.exe
    - sqlite3.exe
    - sqlite3_analyzer.exe

3. Vá para o prompt de comando digitando “CMD” no menu iniciar windows;
4. Entre na pasta que contém os arquivos;
    - Exemplo, para entrar na pasta “c:\diretorio” digite: **cd c:\diretorio**
5. Dentro da pasta que contém os arquivos do SQLite, digite “sqlite3 primeiroBanco”;
6. Neste momento você está na linha de comando do SQLite;
7. Ao passar o nome do banco como parâmetro “primeiroBanco”, você já o cria;
8. Digite .help para obter outras opções do SQLite

Na Figura 34 é possível verificar o SQLite funcionando e pronto para receber comandos em SQL.

```
Administrador: Prompt de Comando - sqlite3
04/07/2019  19:33           805.638 sqlite-dll-win64-x64-3280000.zip
04/07/2019  19:35    <DIR>          sqlite-tools-win32-x86-3280000
04/07/2019  19:34         1.785.665 sqlite-tools-win32-x86-3280000.zip
               8 arquivo(s)    394.871.228 bytes
               3 pasta(s)   128.081.027.072 bytes disponíveis

C:\Users\aead\Downloads>cd sqlite-tools-win32-x86-3280000

C:\Users\aead\Downloads\sqlite-tools-win32-x86-3280000>dir
 O volume na unidade C não tem nome.
 O Número de Série do Volume é F4D2-4411

 Pasta de C:\Users\aead\Downloads\sqlite-tools-win32-x86-3280000

04/07/2019  19:35    <DIR>          .
04/07/2019  19:35    <DIR>          ..
17/04/2019  03:07           498.176 sqldiff.exe
16/04/2019  21:40             6.010 sqlite3.def
16/04/2019  21:40         1.897.472 sqlite3.dll
17/04/2019  03:08           928.768 sqlite3.exe
17/04/2019  03:07         2.012.672 sqlite3_analyzer.exe
               5 arquivo(s)      5.343.098 bytes
               2 pasta(s)   128.081.027.072 bytes disponíveis

C:\Users\aead\Downloads\sqlite-tools-win32-x86-3280000>sqlite3
SQLite version 3.28.0 2019-04-16 19:49:53
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open Banco1
sqlite> .databases
main: C:\Users\aead\Downloads\sqlite-tools-win32-x86-3280000\Banco1
sqlite> _
```

Figura 34 - Console SQLite.
Fonte: O Autor.
Descrição: Nesta figura é possível verificar o acesso à pasta via console do windows, desde a pasta de Download até o *prompt* do banco de dados SQLite.

3.1.2 Instalação do PostgreSQL

Para instalar o PostgreSQL, siga os seguintes passos:

1. Acesse o link:
    - (64bits): https://www.enterprisedb.com/thank-you-downloading-postgresql?anid=1256386
    - (32bits): https://www.enterprisedb.com/thank-you-downloading-postgresql?anid=1256731
2. No sistema Windows, salve e execute o arquivo clicando duas vezes sobre ele;
3. Para instalar, clique sempre em "próximo" sem modificar as configurações, com exceção do campo de senha que você deve inserir uma de sua escolha;
4. OBS: escolha uma senha fácil, apenas para fins didático;
5. Após a instalação, verifique se um serviço de sistema chamado "postgresql-x64-9.6 - PostgreSQL Server 9.6" está iniciado com o status "Em Execução". Para isso, digite "Serviço" no menu iniciar do windows e procure o serviço citado;
6. Após a instalação e o SGBD funcionando, instale o pgAdmin no link: https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v4.10/windows/pgadmin4-4.10-x86.exe. O pgAdmin é uma aplicação Open Source para administrar os bancos de dados do Postgres. A versão atual (4.10) foi desenvolvida para rodar em navegadores de internet, tornando o ambiente multiplataforma.
7. Para iniciar o pgAdmin clique no botão iniciar do Windows e digite pgAdmin depois tecle Enter, o pgAdmin será iniciado em seu *browser* padrão (Chrome, Internet Explorer ou Firefox);
8. No pgAdmin, selecione o servidor ("Servers/PostgreSQL 9.6/Databases/postgres") e no menu "Tools" escolha "Query Tool". Esta ferramenta "Query Tool" serve para executar os scripts SQL. Observe na Figura 35 este ambiente. Pronto!

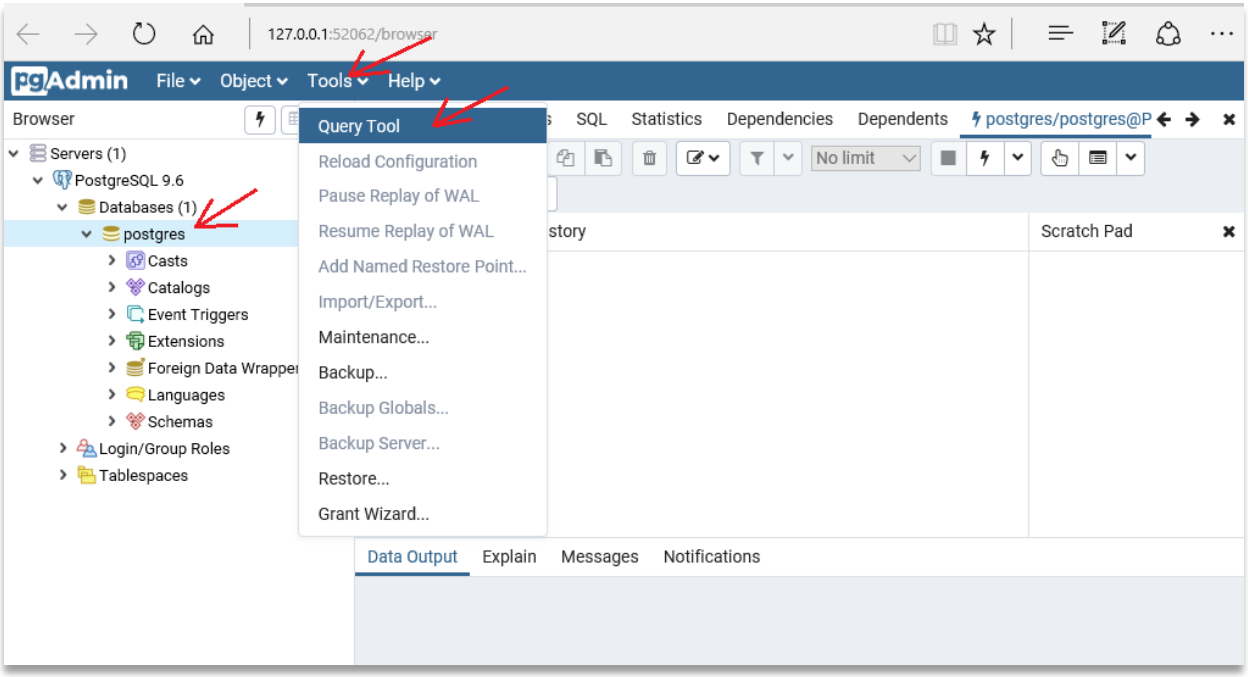

Figura 35 - PgAdmin
Fonte: O Autor.
Descrição: Figura contém a primeira tela do PgAdmin, que é a ferramenta de administração e uso do banco de dados PostgreSQL.

Caso você não esteja conseguindo instalar o SGBD Postgres em sua máquina, ou se você preferir utilizar um ambiente mais prático e didático, você pode utilizar o SQLite, que é uma opção mais leve. Seja qual for o SGBD que você esteja manipulando, os comandos que veremos funcionarão em ambos (com exceção do tipo TIMESTAMP na tabela Venda).

3.2 Scripts do Banco de dados Vendas

3.2.1 Scripts SQL-DDL-CREATE, BD Vendas

Para criar o banco de dados Vendas (visto na primeira competência) em um SGBD, execute os comandos SQL-DDL listados a seguir. Estes comandos estão em formato texto com o objetivo de facilitar a criação do ambiente, e são os mesmos scripts vistos durante todo o curso, cujo modelo conceitual encontra-se na Figura 16. Para isso, abra um dos SGBD´s citados acima, copie os scripts descritos a seguir, cole-os no ambiente de execução e execute-os.

Script SQL-DDL Banco de Dados Vendas (**INÍCIO**) ----

```
CREATE TABLE CLIENTE (
        PKCPF VARCHAR(15) PRIMARY KEY,
        NOME VARCHAR(50) NOT NULL,
        EMAIL VARCHAR(20)
        );

CREATE TABLE TELEFONE(
        PKTELEFONE INTEGER PRIMARY KEY,
        CPF VARCHAR(15),
        TELEFONE VARCHAR(15) NOT NULL,
        FOREIGN KEY (CPF) REFERENCES CLIENTE (PKCPF)
        );
CREATE TABLE ENDERECO (
        PKENDERECO INTEGER PRIMARY KEY,
        CPF VARCHAR(15),
        RUA_NUMERO VARCHAR(50) NOT NULL,
        BAIRRO VARCHAR(30) NOT NULL,
        MUNICIPIO VARCHAR(30) NOT NULL,
        ESTADO CHAR(2),
        CEP VARCHAR(10),
        FOREIGN KEY (CPF) REFERENCES CLIENTE (PKCPF)
        );

CREATE TABLE LOJA (
        PKLOJA INTEGER PRIMARY KEY,
        DESCRICAO VARCHAR(50)
        );

CREATE TABLE PRODUTO (
        PKPRODUTO INTEGER PRIMARY KEY,
        DESCRICAO VARCHAR(50) NOT NULL,
        QTD_ESTOQUE INTEGER NOT NULL,
        UNIDADE_ARMAZENADA VARCHAR(20),
        FKLOJA INTEGER,
        VALOR_UNIDADE DECIMAL(10,2),
        FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA)
        );

CREATE TABLE VENDEDOR (
        PKCPF VARCHAR(15) PRIMARY KEY,
        NOME VARCHAR(50),
        FKLOJA INTEGER,
        FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA)
        );
```

```
CREATE TABLE VENDA (
        PKVENDA INTEGER PRIMARY KEY,
        DATA_HORA DATETIME, --OBS: no postgres coloque o tipo TIMESTAMP
        VALOR DECIMAL(10,2),
        FKCLIENTE VARCHAR(15),
        FKLOJA INTEGER,
        FKVENDEDOR VARCHAR(15),
        FOREIGN KEY (FKCLIENTE) REFERENCES CLIENTE (PKCPF),
        FOREIGN KEY (FKLOJA) REFERENCES LOJA (PKLOJA),
        FOREIGN KEY (FKVENDEDOR) REFERENCES VENDEDOR (PKCPF)
        );
CREATE TABLE VENDA_PRODUTO (
        FKVENDA INTEGER,
        FKPRODUTO INTEGER,
        QUANTIDADE INTEGER,
        PRIMARY KEY (FKVENDA, FKPRODUTO),
        FOREIGN KEY (FKVENDA) REFERENCES VENDA (PKVENDA),
        FOREIGN KEY (FKPRODUTO) REFERENCES PRODUTO (PKPRODUTO)
        );
```

Script SQL-DDL Banco de Dados Vendas (**FIM**) ----

Após executar estes scripts, verifique se as tabelas foram realmente criadas. Para isso, execute o comando ".tables" no SQLite ou "Select * from pg_tables" no SGBD PostgreSQL. O próximo passo é popular estas tabelas e consultar os dados.

3.2.2 Scripts SQL-DML-INSERT, BD Vendas

Com o banco de dados Vendas já criado, você precisa inserir dados nas tabelas. Para isso, copie e execute os scripts SQL-DML-INSERT para cada tabela a seguir:

Tabela Cliente (**INÍCIO**):

```
INSERT INTO CLIENTE VALUES ('008.845.966-52', 'JOÃO DA COSTA', 'JOAODACOSTA@GML.COM');
INSERT INTO CLIENTE VALUES ('885.665.965-78', 'FULANO DA SILVA', 'FULANOSILVA@GML.COM');
INSERT INTO CLIENTE VALUES ('554.854.547-85', 'JOSÉ DAS CANDOCA', 'DASCANDOCA@GML.COM');
INSERT INTO CLIENTE VALUES ('552.312.689-96', 'MARIA DO ROSÁRIO', 'MARIAROSA@GML.COM');
INSERT INTO CLIENTE VALUES ('785.645.589-21', 'FÁTIMA BERNARDO', 'FATIMABERNA@HOT.COM');
```

Tabela Cliente (**FIM**).

Tabela Telefone (**INÍCIO**):

```
INSERT INTO TELEFONE VALUES (1, '008.845.966-52', ‘819965588xx’);
INSERT INTO TELEFONE VALUES (2, '885.665.965-78', ‘819965544xx);
INSERT INTO TELEFONE VALUES (3, '554.854.547-85', ‘819978855xx’);
INSERT INTO TELEFONE VALUES (4, '554.854.547-85', ‘219954896xx’);
INSERT INTO TELEFONE VALUES (5, '552.312.689-96', ‘11213562xx’);
INSERT INTO TELEFONE VALUES (6, '785.645.589-21', ‘85998745xx’);
```

Tabela Telefone (**FIM**).

Tabela Endereço (**INÍCIO**):

```
INSERT INTO ENDERECO VALUES (1, '008.845.966-52', 'RUA R5, NUMERO 43', 'ALTO DO MANDÚ', 'RECIFE', 'PE', '54280656');
INSERT INTO ENDERECO VALUES (2, '008.845.966-52', 'RUA JOSÉ ANTONIO, NUMERO 15', 'VARADOURO', 'OLINDA', 'PE', '53245555');
INSERT INTO ENDERECO VALUES (3, '885.665.965-78', 'RUA B, NUMERO 23', 'ALTO JOSÉ BONIFÁCIO', 'RECIFE', 'PE', '54256666');
INSERT INTO ENDERECO VALUES (4, '885.665.965-78', 'RUA VICENTE DA SILVA, NUMERO 72', 'VILA DA FÁBRICA', 'RECIFE', 'PE', '54203566');
INSERT INTO ENDERECO VALUES (5, '554.854.547-85', 'RUA BERNARDO VIEIRA, NUMERO 900', 'BOA VIAGEM', 'RECIFE', 'PE', '54887778');
INSERT INTO ENDERECO VALUES (6, '554.854.547-85', 'AVENIDA GUARARAPES, NUMERO 100', 'BOA VISTA', 'RECIFE', 'PE', '53245555');
INSERT INTO ENDERECO VALUES (7, '552.312.689-96', 'TRAVESSA DO NEGO BOM, NUMERO 28', 'COMUNIDADE DA COVA', 'OLINDA', 'PE', '52212233');
INSERT INTO ENDERECO VALUES (8, '552.312.689-96', 'RUA FELIX DE BRITO, NUMERO 38', 'CENTRO', 'OLINDA', 'PE', '54215689');
INSERT INTO ENDERECO VALUES (9, '785.645.589-21', 'AVENIDA POTIGUAR, NUMERO 902', 'PIEDADE', 'JABOATÃO DOS GUARARAPES', 'PE', '54280564');
INSERT INTO ENDERECO VALUES (10, '785.645.589-21', 'RUA DO FOGO, NUMERO 34', 'CANDEIAS', 'JABOATÃO DOS GUARARAPES', 'PE', '54879633');
```

Tabela Endereço (**FIM**).

Tabela Loja (**INÍCIO**):

```
INSERT INTO LOJA VALUES (1, 'LOJA PRINCIPAL DO CENTRO DE JABOATÃO');
INSERT INTO LOJA VALUES (2, 'FILIAL JABOATÃO');
INSERT INTO LOJA VALUES (3, 'LOJA PRINCIPAL DO CENTRO DE RECIFE');
INSERT INTO LOJA VALUES (4, 'FILIAL RECIFE RUA DA PALMA');
INSERT INTO LOJA VALUES (5, 'FILIAL RECIFE RUA DO IMPERADOR');
INSERT INTO LOJA VALUES (6, 'LOJA PRINCIPAL DO CENTRO DE OLINDA');
INSERT INTO LOJA VALUES (7, 'FILIAL OLINDA RUA DO SOL');
INSERT INTO LOJA VALUES (8, 'FILIAL OLINDA ALTO DA SÉ');
```

Tabela Loja (**FIM**).

Tabela Produto (**INÍCIO**):

```
INSERT INTO PRODUTO VALUES (1, 'AMACIANTE OMU', 40, 'PEÇA', 1, 12.00);
INSERT INTO PRODUTO VALUES (2, 'AMACIANTE OMU', 50, 'PEÇA', 2, 12.00);
INSERT INTO PRODUTO VALUES (3, 'AMACIANTE OMU', 100, 'PEÇA', 3, 12.00);
INSERT INTO PRODUTO VALUES (4, 'SABONETE DUVI', 520, 'PACOTE C 10', 4, 31.50);
INSERT INTO PRODUTO VALUES (5, 'SABONETE DUVI', 820, 'PACOTE C 10', 5, 31.50);
INSERT INTO PRODUTO VALUES (6, 'SABONETE DUVI', 745, 'PACOTE C 10', 6, 31.50);
INSERT INTO PRODUTO VALUES (7, 'ARROZ TIO JOSÉ', 310, 'SACA C 50KG', 7, 59.90);
INSERT INTO PRODUTO VALUES (8, 'ARROZ TIO JOSÉ', 350, 'SACA C 50KG', 8, 59.90);
INSERT INTO PRODUTO VALUES (9, 'ARROZ TIO JOSÉ', 430, 'SACA C 50KG', 1, 59.90);
INSERT INTO PRODUTO VALUES (10, 'FUBÁ VITAMILHO', 256, 'SACA C 20KG', 2, 25.60);
INSERT INTO PRODUTO VALUES (11, 'FUBÁ VITAMILHO', 542, 'SACA C 20KG', 3, 25.60);
INSERT INTO PRODUTO VALUES (12, 'FUBÁ VITAMILHO', 456, 'SACA C 20KG', 4, 25.60);
```

Tabela Produto (**FIM**).

Tabela Vendedor (**INÍCIO**):

```
INSERT INTO VENDEDOR VALUES ('556.554.998-80', 'JOSÉ PEDRO', 1);
INSERT INTO VENDEDOR VALUES ('539.291.499-02', 'JOÃO CARLOS', 1);
INSERT INTO VENDEDOR VALUES ('778.293.889-20', 'CARLOS RENATO', 2);
INSERT INTO VENDEDOR VALUES ('728.193.490-94', 'FELIPE DE LARA', 3);
INSERT INTO VENDEDOR VALUES ('819.299.300-99', 'JEFERSON DA SILVA', 4);
INSERT INTO VENDEDOR VALUES ('201.237.490-22', 'MARIA JOSÉ', 5 );
INSERT INTO VENDEDOR VALUES ('020.339.495-92', 'DEISE DA CUNHA', 5);
INSERT INTO VENDEDOR VALUES ('234.928.900-04', 'THIAGO DA COSTA', 6);
```

Tabela Vendedor (**FIM**).

Tabela de Venda (**INÍCIO**):

```
INSERT INTO VENDA VALUES (100, '24/05/2019', 105.65, '008.845.966-52', 1, '556.554.998-80');
INSERT INTO VENDA VALUES (101, '20/06/2019', 75.35, '008.845.966-52', 3, '556.554.998-80');
INSERT INTO VENDA VALUES (102, '15/06/2019', 125.20, '885.665.965-78', 5, '539.291.499-02');
INSERT INTO VENDA VALUES (103, '14/07/2019', 353.30, '885.665.965-78', 2, '539.291.499-02');
INSERT INTO VENDA VALUES (104, '10/08/2018', 589.93, '885.665.965-78', 7, '778.293.889-20');
INSERT INTO VENDA VALUES (105, '09/04/2019', 1002.37, '554.854.547-85', 5, '778.293.889-20');
INSERT INTO VENDA VALUES (106, '05/03/2019', 400.67, '554.854.547-85', 8, '778.293.889-20');
INSERT INTO VENDA VALUES (107, '03/11/2019', 316.54, '554.854.547-85', 2, '728.193.490-94');
INSERT INTO VENDA VALUES (108, '30/12/2018', 215.22, '552.312.689-96', 6, '728.193.490-94');
INSERT INTO VENDA VALUES (109, '27/09/2018', 21.88, '552.312.689-96', 2, '819.299.300-99');
INSERT INTO VENDA VALUES (110, '25/10/2018', 659.80, '552.312.689-96', 3, '201.237.490-22');
INSERT INTO VENDA VALUES (111, '20/09/2018', 578.40, '785.645.589-21', 7, '020.339.495-92');
INSERT INTO VENDA VALUES (112, '19/05/2019', 321.70, '785.645.589-21', 8, '020.339.495-92');
```

Tabela de Venda (**FIM**).

Tabela Venda_Produto (**INÍCIO**):

```
INSERT INTO VENDA_PRODUTO VALUES (112, 1, 5);
INSERT INTO VENDA_PRODUTO VALUES (112, 2, 10);
INSERT INTO VENDA_PRODUTO VALUES (112, 3, 20);
INSERT INTO VENDA_PRODUTO VALUES (112, 4, 10);
INSERT INTO VENDA_PRODUTO VALUES (112, 5, 30);
INSERT INTO VENDA_PRODUTO VALUES (112, 6, 15);
INSERT INTO VENDA_PRODUTO VALUES (112, 7, 8);
INSERT INTO VENDA_PRODUTO VALUES (111, 7, 5);
INSERT INTO VENDA_PRODUTO VALUES (111, 6, 10);
INSERT INTO VENDA_PRODUTO VALUES (111, 5, 20);
INSERT INTO VENDA_PRODUTO VALUES (111, 4, 10);
INSERT INTO VENDA_PRODUTO VALUES (111, 3, 30);
INSERT INTO VENDA_PRODUTO VALUES (111, 2, 15);
INSERT INTO VENDA_PRODUTO VALUES (111, 1, 8);
INSERT INTO VENDA_PRODUTO VALUES (110, 3, 5);
INSERT INTO VENDA_PRODUTO VALUES (110, 4, 10);
INSERT INTO VENDA_PRODUTO VALUES (110, 5, 20);
INSERT INTO VENDA_PRODUTO VALUES (110, 6, 10);
INSERT INTO VENDA_PRODUTO VALUES (110, 7, 30);
INSERT INTO VENDA_PRODUTO VALUES (110, 8, 15);
INSERT INTO VENDA_PRODUTO VALUES (110, 9, 8);
INSERT INTO VENDA_PRODUTO VALUES (109, 4, 2);
INSERT INTO VENDA_PRODUTO VALUES (109, 3, 5);
INSERT INTO VENDA_PRODUTO VALUES (109, 2, 22);
INSERT INTO VENDA_PRODUTO VALUES (109, 5, 44);
INSERT INTO VENDA_PRODUTO VALUES (109, 7, 50);
INSERT INTO VENDA_PRODUTO VALUES (109, 6, 23);
INSERT INTO VENDA_PRODUTO VALUES (109, 8, 8);
INSERT INTO VENDA_PRODUTO VALUES (108, 3, 5);
INSERT INTO VENDA_PRODUTO VALUES (108, 4, 10);
INSERT INTO VENDA_PRODUTO VALUES (108, 5, 20);
INSERT INTO VENDA_PRODUTO VALUES (108, 6, 10);
INSERT INTO VENDA_PRODUTO VALUES (108, 7, 30);
INSERT INTO VENDA_PRODUTO VALUES (108, 8, 15);
INSERT INTO VENDA_PRODUTO VALUES (108, 9, 8);
INSERT INTO VENDA_PRODUTO VALUES (107, 5, 8);
INSERT INTO VENDA_PRODUTO VALUES (107, 3, 10);
INSERT INTO VENDA_PRODUTO VALUES (107, 8, 11);
INSERT INTO VENDA_PRODUTO VALUES (107, 9, 13);
INSERT INTO VENDA_PRODUTO VALUES (107, 4, 2);
INSERT INTO VENDA_PRODUTO VALUES (107, 2, 1);
INSERT INTO VENDA_PRODUTO VALUES (107, 1, 6);
INSERT INTO VENDA_PRODUTO VALUES (106, 1, 8);
INSERT INTO VENDA_PRODUTO VALUES (106, 3, 10);
INSERT INTO VENDA_PRODUTO VALUES (106, 4, 11);
INSERT INTO VENDA_PRODUTO VALUES (106, 5, 13);
```

```
INSERT INTO VENDA_PRODUTO VALUES (106, 2, 2);
INSERT INTO VENDA_PRODUTO VALUES (106, 7, 1);
INSERT INTO VENDA_PRODUTO VALUES (106, 9, 6);
INSERT INTO VENDA_PRODUTO VALUES (105, 9, 1);
INSERT INTO VENDA_PRODUTO VALUES (105, 7, 1);
INSERT INTO VENDA_PRODUTO VALUES (105, 8, 3);
INSERT INTO VENDA_PRODUTO VALUES (105, 6, 5);
INSERT INTO VENDA_PRODUTO VALUES (105, 4, 6);
INSERT INTO VENDA_PRODUTO VALUES (105, 5, 10);
INSERT INTO VENDA_PRODUTO VALUES (105, 3, 1);
INSERT INTO VENDA_PRODUTO VALUES (104, 1, 2);
INSERT INTO VENDA_PRODUTO VALUES (104, 2, 2);
INSERT INTO VENDA_PRODUTO VALUES (104, 3, 5);
INSERT INTO VENDA_PRODUTO VALUES (104, 4, 7);
INSERT INTO VENDA_PRODUTO VALUES (104, 9, 10);
INSERT INTO VENDA_PRODUTO VALUES (104, 8, 13);
INSERT INTO VENDA_PRODUTO VALUES (104, 7, 20);
INSERT INTO VENDA_PRODUTO VALUES (103, 6, 1);
INSERT INTO VENDA_PRODUTO VALUES (103, 4, 5);
INSERT INTO VENDA_PRODUTO VALUES (103, 3, 3);
INSERT INTO VENDA_PRODUTO VALUES (103, 2, 8);
INSERT INTO VENDA_PRODUTO VALUES (103, 1, 10);
INSERT INTO VENDA_PRODUTO VALUES (103, 7, 12);
INSERT INTO VENDA_PRODUTO VALUES (103, 9, 2);
INSERT INTO VENDA_PRODUTO VALUES (102, 9, 20);
INSERT INTO VENDA_PRODUTO VALUES (102, 8, 10);
INSERT INTO VENDA_PRODUTO VALUES (102, 7, 7);
INSERT INTO VENDA_PRODUTO VALUES (102, 1, 8);
INSERT INTO VENDA_PRODUTO VALUES (102, 2, 15);
INSERT INTO VENDA_PRODUTO VALUES (102, 3, 18);
INSERT INTO VENDA_PRODUTO VALUES (102, 4, 4);
INSERT INTO VENDA_PRODUTO VALUES (101, 3, 1);
INSERT INTO VENDA_PRODUTO VALUES (101, 1, 1);
INSERT INTO VENDA_PRODUTO VALUES (101, 9, 3);
INSERT INTO VENDA_PRODUTO VALUES (101, 7, 5);
INSERT INTO VENDA_PRODUTO VALUES (101, 5, 6);
INSERT INTO VENDA_PRODUTO VALUES (101, 4, 10);
INSERT INTO VENDA_PRODUTO VALUES (101, 2, 1);
INSERT INTO VENDA_PRODUTO VALUES (100, 4, 3);
INSERT INTO VENDA_PRODUTO VALUES (100, 6, 2);
INSERT INTO VENDA_PRODUTO VALUES (100, 8, 5);
INSERT INTO VENDA_PRODUTO VALUES (100, 2, 4);
INSERT INTO VENDA_PRODUTO VALUES (100, 3, 1);
INSERT INTO VENDA_PRODUTO VALUES (100, 7, 5);
INSERT INTO VENDA_PRODUTO VALUES (100, 9, 7);
```

Tabela Venda_Produto(**FIM**).

É importante observar que, caso você queira inserir mais registros nas tabelas do banco de dados Vendas e incrementar o conjunto de dados definidos acima (sugiro que você faça isso), deve ficar atento às chaves primárias dos dados que já foram inseridos, uma vez que não é possível inserir chaves repetidas. Caso você tente inserir um registro com a chave primária repetida, o SGBD deve exibir uma mensagem de erro, informando que você tentou violar uma regra do banco de dados que é a duplicidade de chaves primárias. Tente inserir outros registros na tabela Venda_Produto, pois inserimos apenas três tipos de produtos, a saber: Amaciante OMU, Arroz Tio José e Sabonete Duvi. Tente variar os produtos, levando em consideração o estoque em cada loja.

A seguir veremos scripts SQL-DML-SELECT, com funções de agregação e junção entre tabelas.

3.2.3 Scripts SQL-DML-SELECT, BD Vendas

Com as tabelas criadas e os dados já inseridos, podemos então realizar consultas neste banco de dados. As consultas em banco de dados são realizadas pelo comando SELECT e suas variações. Estas variações vão depender da informação que o usuário quer obter. Por exemplo, em nosso banco de dados Vendas, o usuário pode querer saber uma das seguintes perguntas: quantos produtos foram vendidos em determinado dia? Quais são os produtos mais vendidos no mês? Qual a loja que mais vende? Qual o vendedor que mais vende? E assim por diante. A seguir temos alguns exemplos de consultas ao banco de dados Vendas.

A forma mais comum do *script* SELECT é: **select * from produto**. Neste script, como já vimos, têm-se as palavras reservadas “SELECT e FROM”. O asterisco significa que todas as colunas da referida tabela serão retornadas. Se o usuário desejar especificar qual(ais) coluna(s) deseja visualizar, ele deve substituir o asterisco pelo nome da coluna. Por exemplo: **select descricao, qtd_estoque from produto**; retornando, neste caso, a descrição do produto e sua quantidade em estoque. Caso o usuário queira retornar apenas um registro específico, deve-se utilizar a cláusula WHERE, que insere uma condição a ser satisfeita. Por exemplo, a consulta a seguir retorna apenas a descrição e quantidade em estoque do produto “Amaciante Omu”: **select descricao, qtd_estoque from produto where descricao = “AMACIANTE OMU”.** Neste último comando utilizamos o operador de comparação igual “=”. No entanto, podemos utilizar alguns outros operadores mais comuns:

diferença “<>”, maior que “>”, menor que “<”, maior ou igual “>=”, menor ou igual “<=”; como também outros mais específicos: intervalo “BETWEEN”, dentro “IN” e contém “LIKE”. O operador “BETWEEN” funciona da seguinte forma: **select * from venda_produto where quantidade between 2 and 5;** onde serão retornados registros que possuem quantidade vendida entre 2 e 5. De forma similar, o operador “IN” funciona da seguinte maneira: **select * from venda_produto where quantidade in (2, 5, 7);** onde serão retornados os registros que possuem as quantidades especificadas dentro de um conjunto. Por último, não menos importante, temos o operador “LIKE”, que funciona da seguinte maneira: **select * from cliente where nome like "MARIA%";** onde serão retornados todos os registros cujos nomes possuem a inicial “MARIA”. O símbolo de percentual “%” significa dizer que não importa o que vem após, neste exemplo. Caso fosse necessário retornar todos os nomes de clientes que contêm o nome MARIA, quer seja no início, no meio ou no fim, colocaríamos o símbolo de percentual antes e depois, assim: “...nome like “%MARIA%””; ou caso quisesse que o nome do cliente terminasse com MARIA, seria assim: “...nome like “%MARIA””. Simples, não?

Uma variação do script SELECT simples (visto acima) é a utilização de funções de agregação. Estas funções são cálculos simples feitos em colunas numéricas. As mais comuns são: soma, média, mínimo, máximo, contagem; sujas sintaxes são, respectivamente: SUM, AVG, MIN, MAX, COUNT. Um exemplo simples é somar a quantidade total vendida dos produtos: **select sum(quantidade) from venda_produto,** ou querer saber a quantidade de registros da tabela produto: **select count(*) from produto**. Embora estes exemplos sejam bastantes intuitivos e de fácil assimilação, em um ambiente corporativo as consultas têm necessidades diversas que vão depender do negócio e são mais complexas. Sendo assim, contar os registros em uma tabela não têm muita informação do negócio, não é mesmo? Portanto, visando atrelar as consultas de soma e count ao negócio do banco de dados Vendas, vamos retornar a soma dos produtos vendidos, agrupando pelo identificador FKPRODUTO, da tabela VENDA_PRODUTO: **select fkproduto, sum(quantidade) from venda_produto group by fkproduto.** Observe que neste script há a especificação das palavras reservadas “GROUP BY” junto à coluna que você quer agrupar FKPRODUTO, ao final do comando. O “GROUP BY” define que você quer agrupar por categoria uma função de agregação, que neste caso é a função de soma. A execução deste script no banco de dados SQLite é visto na Figura 36. Para melhorar ainda mais nossa consulta, vamos inserir uma ordenação ao conjunto de dados retornados, utilizando a palavra reservada ORDER BY, que pode ser visto na Figura 37.

```
sqlite> select fkproduto, sum(quantidade) from venda_produto group by fkproduto;
1|48
2|80
3|114
4|90
5|171
6|76
7|174
8|80
9|78
```

Figura 36 - Comando SQL-DML-SELECT com função de agregação Soma.
Fonte: O Autor.
Descrição: Nesta figura é possível verificar um comando SELECT com uma função de agregação Soma.

Observe na Figura 36 que, para cada identificador do produto tem uma quantidade, informando que, por exemplo, foram vendidos 48 produtos com o identificador 1.

```
sqlite> select fkproduto as fkproduto, sum(quantidade) soma from venda_produto group by fkproduto order by sum(quantidade);
1|48
6|76
9|78
2|80
8|80
4|90
3|114
5|171
7|174
sqlite>
```

Figura 37 - Comando SQL-DML-SELECT com função de agregação Soma e função de ordenação.
Fonte: O Autor.
Descrição: Nesta figura é possível verificar um comando SELECT com uma função de agregação Soma ordenada.

Observa-se na Figura 37 que a coluna retornada da soma foi ordenada. Caso o usuário queira ordenar por outra coluna, deve inserir logo após das palavras reservadas “ORDER BY”. Caso o usuário queira que os dados sejam ordenados na forma inversa, ou seja, maiores primeiro, deve-se inserir a palavra reservada “DESC” logo após a coluna agrupada, ou seja “...ORDER BY COLUNA DESC”. Tente isso em seu ambiente.

Embora você já esteja retornando consultas em um SGBD, o que é de grande importância (parabéns por isso!), você vai concordar comigo que não temos muita informação na prática. Na Figura 37 por exemplo, não sabemos qual produto é representado pelo identificador 1 (na primeira linha). Para que a gente consiga exibir a descrição do produto, teremos que buscar esta informação na tabela PRODUTO, uma vez que na tabela VENDA_PRODUTO só existem colunas numéricas. Para isso utilizaremos o conceito de junção entre tabelas, onde duas tabelas estão relacionadas por um

identificador em comum. Sendo assim, utilizaremos as palavras reservadas “INNER JOIN” e “ON”, cujo comando exemplo é: **select * from venda_produto vp inner join produto p on p.pkproduto = vp.fkproduto**. Parece complexo, não? Mas mantenha a calma, você está prestes a dominar consultas em bancos de dados. Este comando vincula a tabela VENDA_PRODUTO com a tabela PRODUTO. Temos também a presença de **ALIAS,** que são “apelidos” de tabelas ou colunas; neste caso demos o alias “VP” para a tabela VENDA_PRODUTO e “P” para PRODUTO, evitando a repetição do respectivo nome. O INNER JOIN informa que você quer juntar duas tabelas de forma que a condição “ON” seja atendida, neste caso as chaves PKPRODUTO da tabela PRODUTO deve ser igual a FKPRODUTO da tabela VENDA_PRODUTO. A seguir veremos os tipos de junções.

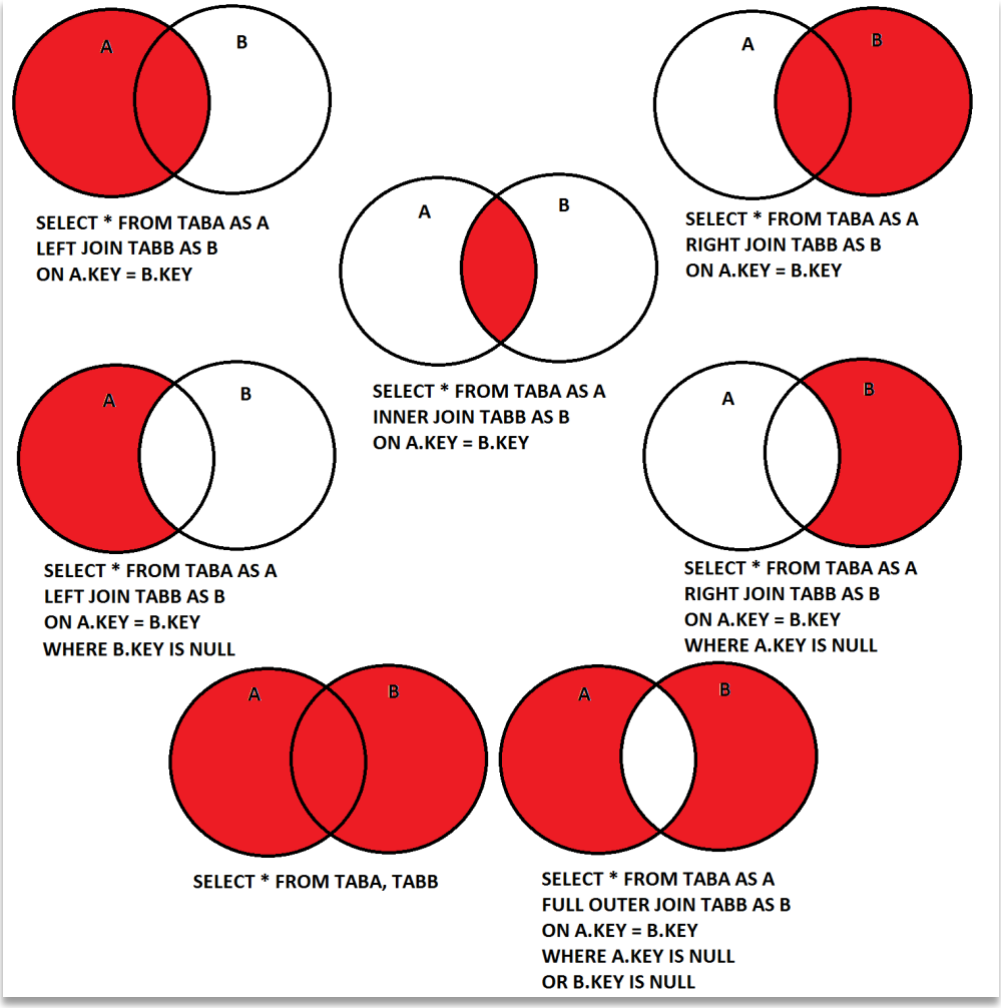

Figura 38 - Junção entre Tabelas.
Fonte: O Autor.
Descrição: Esta figura contém os diferentes tipos de junções entre tabelas.

Existem outras formas de juntar tabelas, que podem ser usadas de acordo com a necessidade da informação a ser retornada. O conceito de juntar tabelas é definido por meio de operações de conjuntos, onde podemos ter as operações: CROSS JOIN, LEFT JOIN, RIGHT JOIN, INNER JOIN e FULL OUTER JOIN. Na Figura 38 é possível verificar estas relações com exemplos em SQL.

Observe na Figura 38 os conjuntos A e B, cuja informação a ser extraída está representada na cor vermelha. Para a operação LEFT JOIN, os dados da esquerda são retornados. Na prática, significa dizer que, quando não houver correspondente em ambas tabelas, isto é, em apenas uma delas, serão retornados apenas os dados da tabela da esquerda. Por exemplo, se na tabela A existir um registro que não existe correspondente em B, mesmo sem ter relação em comum, os registros do lado A serão retornados, com o lado de B nulo. O mesmo se dá para a operação RIGHT JOIN (o SGBD SQLite não suporta as operações RIGHT JOIN e FULL OUTER JOIN), que é a operação LEFT JOIN invertendo as tabelas. A operação INNER JOIN é a mais usual, uma vez que corresponde aos registros que satisfazem à condição entre chaves, sendo os registros que não possuem relação, ignorados. A operação CROSS JOIN é um produto cartesiano (operação de multiplicação), onde cada registro da tabela A se relaciona com cada registro da tabela B, sem que seja necessário uma condição satisfeita (o CROSS JOIN é um FULL OUTER JOIN sem condição). Ou seja, se temos 10 registros na tabela A, mais 10 registros na tabela B, teremos um total de 100 registros retornados, no CROSS JOIN. Por fim, tem-se a operação FULL OUTER JOIN, que retorna o cruzamento de ambas tabelas levando em consideração uma condição entre elas.

Para você entender melhor as operações de junção entre tabelas, sugiro que você faça testes em seu ambiente de desenvolvimento, inserindo registros aleatórios e entendendo na prática os conceitos aqui observados. Insira registro que estejam em uma tabela, mas não em outra, para que quando você utilizar as operações de junção você consiga identificar as variâncias.

3.2.4 Scripts SQL-DML-UPDATE, BD Vendas

Como já foi visto, a operação de atualização de dados é feita pelo comando UPDATE-SET-WHERE. Este comando deve ser observado com muita atenção, uma vez que você vai realizar uma alteração nos dados e pode ser danoso. Sugiro, antes de realizar a atualização, verifique se há algum banco de dados de “teste” junto ao administrador de bancos de dados, e evite realizar este tipo de operação em bancos de dados em produção.

Para evitar erros, primeiramente, realize uma operação de consulta, especificando a cláusula “WHERE” para retornar o(s) valor(es) que você deseja alterar. Como um exemplo, digamos que houve uma venda de três (3) unidades do produto “AMACIANTE OMU” da loja 1. Efetue um comando de “SELECT” retornando o produto que contenha a descrição mencionada, cuja loja contenha o identificador “1”: **select * from produto where descricao = "AMACIANTE OMU" and FKLOJA = 1;**. Se você realizou as operações de “INSERT” na criação do banco de dados Vendas, esta consulta retornará o seguinte resultado: PK_PRODUTO=1 | DESCRICAO=AMACIANTE OMU | QTD_ESTOQUE=40 | UNIDADE_ARMAZENADA=PEÇA | FK_LOJA=1 | VALOR_UNIDADE=12. Então, para alterar a quantidade, de forma que seja diminuída em três peças, você deve realizar o seguinte comando: **update produto set qtd_estoque=37 where descricao = "AMACIANTE OMU" and FKLOJA = 1;**. Observe que os dois comandos citados possuem a mesma cláusula “WHERE”, impedindo que outros registros sejam alterados de forma indesejável.

É interessante também, em vez de informar o novo valor já calculado, você buscar o valor atual da quantidade por meio de uma consulta (conhecido como Sub-Select, ou Sub-consulta), subtrair pela quantidade vendida, e obter o novo resultado dinamicamente (Figura 39). Os comandos que contêm Sub-Select´s são executados após as Sub-Rotinas terminarem. Sempre o comando interno é executado primeiro que o mais externo.

```
UPDATE PRODUTO
SET QTD_ESTOQUE = ( SELECT QTD_ESTOQUE-5
                    FROM PRODUTO
                    WHERE DESCRICAO = "AMACIANTE OMU"
                    AND FKLOJA = 1)

WHERE DESCRICAO = "AMACIANTE OMU"
AND FKLOJA = 1;
```

Figura 39 - Comando UPDATE com Sub-Select.
Fonte: O Autor.
Descrição: Comando de atualização de dados ‘UPDATE’ contendo um sub-select, onde é retornado o valor atual e subtraído em 5 a quantidade em estoque.

A Figura 39 apresenta um exemplo de atualização da tabela Produto do banco de dados Vendas, de forma que o novo valor é calculado dinamicamente utilizando um Sub-Select. Observe que o Sub-Select entre os parênteses retorna a quantidade atual em estoque (QTD_ESTOQUE)

subtraída por cinco. Você deve tentar este comando em seu ambiente de desenvolvimento, alterando o valor subtraído e a loja que o produto pertence.

3.2.5 Scripts SQL-DML-DELETE, BD Vendas

O comando SQL-DELETE possui a mesma observação do comando UPDATE abordado anteriormente: cuidado para não apagar o que você não quer

Este comando EXCLUI registros do banco de dados, obedecendo o filtro da cláusula "WHERE". Como teste, vamos apagar o registro que foi alterado no comando de atualização feito anteriormente na Figura 39. Para isso, utilize o seguinte comando: **delete from produto where descricao = "AMACIANTE OMU" and fkloja = 1;**. É importante observar que, esta tabela contém três registros de produtos com a descrição "AMACIANTE OMU" variando apenas a loja. Caso o comando de DELETE omitisse a segunda condição "AND FKLOJA = 1", os três registros do "AMACIANTE OMU" seriam apagados. Teste isso em seu ambiente.

3.2.6 Scripts SQL-DDL-DROP, BD Vendas

Após cumprido todo o fluxo da informação (definição, consulta e tomada de decisão) em cima do dado, o usuário de banco de dados pode querer excluir a tabela. Observação: preciso dizer que esta operação é de extrema cautela e exceção?

Para excluir uma tabela do banco de dados execute o seguinte comando: **drop table nome_tabela**. Este comando exclui a tabela e os dados, não sendo possível sua recuperação caso não exista um backup.

Para se ter um maior domínio sobre operações de UPDATE, DELETE e DROP; de forma que você evite um verdadeiro desastre, consulte nos materiais complementares os comandos voltados para controle de transações: **COMMIT** e **ROLLBACK**. Como também backups de bancos de dados.

Conclusão

Enfim, terminamos!

A disciplina de banco de dados está diretamente ligada a tópicos avançados, tais como: programação em PL/SQL, administração de banco de dados, criação de bancos de dados não convencionais (bancos de dados ativos, bancos de dados geográficos, etc), modelagem de dados, *data science*, *big data*, *business intelligence*, entre muitos outros. É importante que você se dedique e domine os conceitos abordados nesta disciplina, como um início no mundo dos dados.

Na primeira competência é esperado que você entenda os princípios de bancos de dados relacionais: chaves, estrutura de tabelas, relações, dado, sgbd, banco de dados, e sql.

Na segunda competência, espera-se que você absorva os conceitos de modelagem de bancos utilizando, basicamente, duas abordagens: ER e UML. Como também os conceitos relacionados à modelagem por meio de ferramentas do tipo CASE, projeto de banco de dados passando pelos diferentes níveis de abstração, e transformação para SQL-DDL propriamente dito.

A terceira competência possui uma abordagem prática, exigindo que você elabore um ambiente prático de banco de dados executando *scripts* SQL e suas variações.

Espero que você realize testes em seu ambiente, criando outras tabelas, relacionando-as e consultando os diferentes tipos de dados e informações para qualquer negócio. Após você aprender a linguagem de programação Android (aqui mesmo no curso), tente conectá-lo ao SGBD SQLite e crie um banco de dados com as operações: Consultar, Ler, Atualizar e Apagar (CRUD – Create, Read, Update and Delete). Desta forma você poderá compartilhar uma aplicação funcional para dispositivos *mobiles*.

Muito obrigado pela sua atenção empreendida, e até a próxima!

Referências

ALVES, E., FRANCO, N. M., NASCIMENTO, A., & FIDALGO, R. N. **EERCASE: Uma Ferramenta para Apoiar o Estudo do Projeto Conceitual de Banco de Dados.** In Anais dos Workshops do Congresso Brasileiro de Informática na Educação (Vol. 3, No. 1, p. 98, 2014).

BRAMBILLA, Marco; CABOT, Jordi; WIMMER, Manuel. **Model-driven software engineering in practice.** Synthesis Lectures on Software Engineering, v. 1, n. 1, p. 1-182, 2012.

CHEN, PETER PIN-SHAN. **The Entity-Relationship Model-Toward a Unified View of Data**. ACM Transactions on Database Systems, v. 1, n. 1, p. 9-36, 1976.

ELMASRI, R., NAVATHE , S. B. **Sistemas de banco de dados**. 2005

SARMENTO, J., C. **Uma abordagem MDD para prover integridade topológica e de rede em projeto conceitual de banco de dados espaciais**. Dissertação de Mestrado. Universidade Federal de Pernambuco. 2015.

Minicurrículo do Professor

Jones Cavalcanti Sarmento

Bacharel em Sistemas de Informação pela Universidade de Pernambuco (2012) e mestre em Ciências da Computação pela Universidade Federal de Pernambuco (2015). Tem experiência na área de Ciência da Computação com ênfase em análise, integração, modelagem e desenvolvimento de sistemas envolvendo dados. Tem interesse e já trabalhou com as seguintes tecnologias: Big Data (Spark, Hive, Hadoop, Python, HBase), AWS Cloud (EMR), Modelagem Multidimensional, Ferramentas Olap, Business Intelligence, ETL, Banco de Dados Espaciais, Programação Orientada a Objetos, Sistemas Georreferenciados e SGBD´s. Atualmente trabalha como pesquisador pleno na empresa Intelivix e é professor EAD do curso de Desenvolvimento de Sistemas da Secretaria do Estado de Pernambuco.