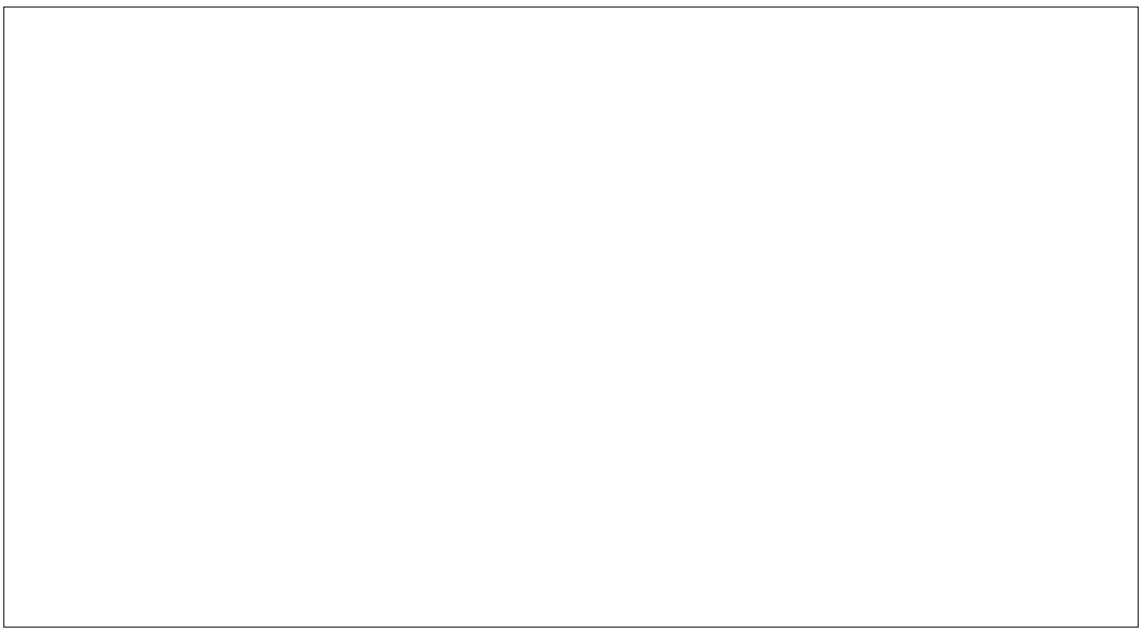
BANCO DE DADOS

ENSINO A DISTÂNCIA

uniAvan
Centro Universitário Avantis



PLANO DE ESTUDOS

EMENTA

Fundamentos de um Sistema Gerenciador de Banco de Dados. Projeto de Banco de Dados: Conceitos, Dependência Funcional, Restrições de Integridade e Formas Normais. SQL: Linguagem de Definição, Manipulação e Controle de Dados. Implementação de um Modelo de Dados. Arquitetura de um Sistema Gerenciador de Banco de Dados.

OBJETIVOS DA DISCIPLINA

- Compreender os conceitos e arquitetura de bancos de dados e Sistemas Gerenciadores de Bancos de Dados -SGBD;
- Aplicar os conceitos de modelagem e banco de dados, entendendo o contexto do projeto e implementação de um banco de dados;
- Aplicar as técnicas de modelagem conceitual de dados e desenvolver uma visão crítica e sistemática sobre a resolução de problemas;
- Entender o conceito de dependências funcionais e aplicá-lo na normalização de dados;
- Conhecer e executar os conceitos das operações da álgebra relacional e sua transcrição para comandos da Linguagem SQL;
- Aplicar a Linguagem SQL nas operações de manipulação de bases de dados relacionais.

O PAPEL DA DISCIPLINA NA FORMAÇÃO DO ACADÊMICO

Prezado(a) Aluno(a),

Em nosso mundo atual, vivenciamos um excesso de informações que nos circunda, nem todas de qualidade comprovada. Enquanto pessoa física, isto já é, em muitos casos, difícil de administrar, então considere a situação de uma empresa qualquer. Diariamente,

são dezenas de milhares de dados (ou milhões?) e informações que passam pelos mais variados setores de qualquer empresa.

Desta forma, vem à tona a principal questão, no que tange ao assunto atual: como armazenar e gerenciar estes dados? A resposta é: os sistemas de gerenciamento de banco de dados. Neste caderno, vamos aprender que sistemas são estes e como eles podem ser utilizados no suporte aos sistemas de informação.

Se olharmos para o recurso como um depósito de informações, poderemos ter a ideia equivocada de que o banco de dados faz somente isso. Se fosse assim, planilhas poderiam ser utilizadas para tal finalidade. Entretanto, o armazenamento das informações representa uma pequena parte do processo todo. É necessário, também, termos mecanismos de controle de acesso, garantia de integridade, persistência e tantos outros mecanismos que garantam a permanente qualidade dos dados. Uma vez armazenados, precisamos, também, de recursos que nos permitam seu acesso e localização, de modo eficaz e eficiente.

De forma simplista, os Sistemas Gerenciadores de Banco de Dados são softwares que permitem que os nossos dados sejam armazenados, organizados, protegidos, atualizados, acrescentados, excluídos e acessados sempre que necessário, devendo corresponder às demandas das aplicações, conforme as necessidades destas. Por conveniência, costumamos chamar os sistemas, abreviados por SGBD, apenas bancos de dados.

Na primeira unidade deste caderno, vamos conhecer a história e a estrutura dos sistemas de gerenciamento de banco de dados, seus princípios de operação e construção, bem como as diferentes organizações que eles apresentam. De forma especial, será apresentado o modelo relacional que representa a estrutura mais comum nos bancos atuais.

Na segunda unidade, estudaremos o projeto de bancos de dados, passando por etapas, como a elaboração de um modelo de armazenamento de informações, sua qualificação enquanto modelo representativo e de que maneira ele será criado em uma base de dados. Analisaremos, também, os diferentes modelos de dados que podem ser criados.

Na unidade seguinte de nosso caderno, trabalhemos com modelagem de dados, ou seja, com o projeto inicial do banco de dados, além das informações que nele deverá conter e por qual motivo.

Para encerramos, compreenderemos a Linguagem SQL. Dificilmente se utilizará uma base de dados, na atualidade, sem que tenhamos conhecimento e acesso aos recursos da linguagem mais popular de manipulação de bancos relacionais que é o SQL, assunto e objeto de estudo da quarta e última unidade.

PROFESSOR

APRESENTAÇÃO DO AUTOR

Aproveito este espaço para me apresentar, chamo-me professor **Marcos César Carrard**. Sou graduado em Ciências da Computação, pela Universidade de Passo Fundo/RS (1990) e especialista de Ciências da Computação, pela Pontifícia Universidade Católica do RS (1991).

Atualmente, sou professor do Centro Universitário Avantis (UNIAVAN), nas disciplinas de programação. Também sou empresário no ramo de desenvolvimento de software. Tenho experiência na área de Sistemas de Informação e Ciências da Computação, principalmente em áreas como: programação, banco de dados, análise e projeto de sistemas, computação gráfica e processamento de imagens.

E-mail: marcos.carrard@avantis.edu.br
Link Lattes: http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4705963J5

SUMÁRIO

unidade 1

SISTEMA GERENCIADOR DE BANCO DE DADOS

1.1 INTRODUÇÃO À UNIDADE

Figura 1 - Necessidade da gestão de dados nas organizações.
Fonte: https://www.capterra.com.br/blog/846/bancos-de-dados-gratuitos-e-de-codigo-aberto/> Acesso em: maio de 2020.

Olá! Seja bem-vindo!

Hoje, somos bombardeados por uma excessiva quantidade de informações, devendo ser o gerenciamento muito completo. Neste contexto, os sistemas gerenciadores de banco de dados (SGBD's) surgem como uma resposta da tecnologia da informação, fornecendo uma arquitetura de software segura e com capacidade de gestão para tamanha quantidade de dados, de forma segura e íntegra.

A informação constitui o bem de maior valia na sociedade atual. Sem ela, as organizações não podem operar, pois é sua matéria-prima básica. Produzir informação é a etapa seguinte do tratamento de dados. Dados são a sua forma bruta, sem significado, enquanto que informação é um dado que ganhou importância e significado dentro de um contexto. As informações são a base para a construção do conhecimento, inclusive organizacional, como também para a sabedoria.

Desta forma, cabe às organizações a tarefa de coletar, organizar, armazenar e recuperar os dados e informações que se produzem. Para tanto, elas fazem uso de SGBD ou, de forma mais simples, Bancos de Dados, ou seja, ferramentas que fazem, de forma correta e apropriada, este gerenciamento.

Na primeira unidade vamos conhecer um pouco da história das ferramentas SGBDs e conhecer seus conceitos fundamentais. Além disso, apresentaremos as principais arquiteturas disponíveis e focaremos na arquitetura mais frequente, nas aplicações da

atualidade, que é o modelo relacional.

Os principais objetivos de aprendizagem desta unidade são: compreender os conceitos e arquitetura de bancos de dados e Sistemas Gerenciadores de Bancos de Dados -SGBD; conhecer e saber a finalidade dos conceitos principais, utilizados na definição de bases relacionais; identificar o modelo relacional de banco de dados e aplicar seus conceitos na construção de bases de dados.

1.2 INTRODUÇÃO

1.2.1 DEFINIÇÃO

Estamos falando, desde o início deste caderno, sobre bancos de dados e suas aplicações. Mas afinal, o que é um Banco de Dados e o que é um Sistema Gerenciador de Banco de Dados? Para efeitos práticos, estes dois termos irão, em nosso caderno, muito em breve, representar a mesma coisa. Entretanto, na teoria eles não são!

Vejamos as definições de dois autores clássicos desta área. Em DATE, 2004, o autor define um banco de dados como:

> *Um banco de dados é uma coleção de dados operacionais armazenados usados pelas aplicações de uma determinada organização* (Date, 2004).

Observe que, nesta definição, o autor enfatiza um ponto chave de todos os bancos de dados: há sempre uma organização. Antes disso, banco de dados é uma coleção de dados armazenados de alguma forma que possibilite o seu uso (operacional...).

Para outros autores que são referência na área, ELMASRI e NAVATHE, 2019, podemos definir bancos de dados como:

> *Um banco de dados é uma coleção de dados relacionados* (Elmasri e Navathe, 2019).

Por mais que a ênfase seja semelhante, os autores apontam outro ponto crucial dos bancos de dados: o relacionamento entre as informações. Um dos princípios básicos do projeto de um banco de dados é a não replicação e redundância entre as informações. Desta forma, é essencial que elas estejam relacionadas. Por exemplo, não devemos armazenar a cidade, CEP e estado de cada cliente junto ao cadastro dele. Recomenda-se que estas

informações sejam armazenadas de forma separada e que o cadastro do cliente aponte (se relacione!) com elas, de forma que não seja necessário repetir o cadastro dos dados de localização inúmeras vezes, produzindo erros e redundância no armazenamento.

Navathe e Elmasri, na mesma obra de 2019, sustentam que bancos de dados têm outras características:

1. Representam algum aspecto do mundo real (minimundo ou universo do discurso);
2. São logicamente coerentes e com algum significado;
3. Projetados, construídos e gerados (povoados) no contexto de alguma aplicação específica.

Por outro lado, o que é um Sistema Gerenciador de Banco de Dados ou SGBD, para simplificar? Os SGBD são sistemas constituídos de um conjunto de programas (software) que permitem criar e manter um banco de dados. Um banco de dados, junto com um sistema que o gerencia (SGBD), constitui o que chamamos de um **Sistema de Banco de Dados**.

Observe que aquilo que chamamos popularmente de banco de dados em nossas aplicações são, em quase todos os casos, a união destes dois elementos. Por exemplo, MySQL, Postgres, Oracle, SQL Server e tantos outros, proprietários ou não, são Sistemas de Banco de Dados, para além de um banco de dados. Ou seja, eles fornecem inúmeros recursos que nos permitem manipular e gerenciar nossos dados, além do armazenamento.

Na Figura 2, além do que foi citado, podemos observar que, no Sistema de Banco de Dados, ainda existe um catálogo de dados, o qual é uma coleção de metadados, isto é, são dados que explicam os dados. Está confuso isso?! Os metadados carregam informações sobre a organização e natureza dos dados armazenados, podendo-se gerenciar mais facilmente as informações armazenadas no banco de dados. Assim, os metadados organizam a informação armazenada, permitindo que o acesso e a manipulação das informações ocorra com mais eficácia e eficiência.

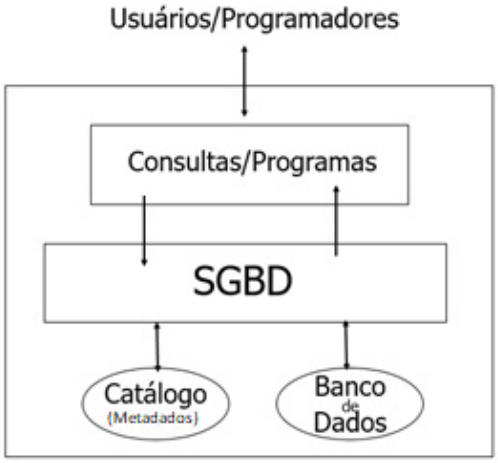


Figura 2 - Sistema de Banco de Dados.
Fonte: Elaborado pelo autor (2020).

1.3 VANTAGENS E DESVANTAGENS

Os sistemas de banco de dados, de forma geral, apresentam algumas características importantes:

- **Autodescrição dos dados:** representada pelo catálogo ou pelos metadados, a informação armazenada contém todas as informações necessárias para interpretar e tratar cada dado armazenado. Desta forma, eles são ditos como autodescritos, ou seja, contêm informações que os descrevem.
- **Isolamento entre programas e dados:** o SGBD coloca uma ou mais camadas entre as aplicações dos usuários e os dados armazenados. Isto garante isolamento dos dados e faz com que o SGBD seja o responsável principal pela integridade deles.
- **Suporte a múltiplas visões:** os sistemas de banco de dados permitem que cada usuário ou aplicação, de acordo com a sua necessidade, tenha uma visão própria e adequada dos dados, sem que possua acesso a coisas que não lhe são pertinentes.
- **Compartilhamento de dados:** os SGBD's permitem que os dados armazenados neles sejam compartilhados e, além disso, fornecem mecanismos que permitem o controle da integridade de cada dado armazenado.
- **Concorrência:** atuando conjuntamente ao compartilhamento, os SGBD implementam o controle de concorrência, de forma automática ou manual, que permite controlar o acesso simultâneo (concorrência) aos mesmos dados armazenados e permitir, mesmo nesta situação, que não ocorra acesso ou alteração indevida aos dados.

Segundo Elmasri e Navathe (2019), existem claras vantagens e problemas no uso de banco de dados. Para discutirmos o assunto, vamos primeiramente analisar as vantagens desta abordagem para armazenamento de informações:

1- Controle de redundância dos dados: o armazenamento de informação, de forma isolada, pelos usuários, tende a gerar redundância, ou seja, vários usuários mantêm cópias diferentes dos mesmos dados. Com o uso de bancos de dados, apenas uma cópia de cada dado pode ser mantida e todos os usuários a acessam em tempo real.

2- Controle de acesso (segurança): banco de dados restringem o acesso não autorizado aos dados, ou a parte deles. Existem diferentes usuários em um banco de dados, e a eles podemos dar visões distintas dos dados. Com isso, além de um usuário não autorizado não conseguir entrar, aqueles que conseguem, podem acessar somente o que é necessário e nada mais. Este é um importante mecanismo de segurança nas aplicações.

3- Armazenamento persistente dos dados: quando programamos em C ou Java,

nossos objetos e estruturas estão em memória principal e deixam de existir, assim que o programa finaliza, ou quando saímos da máquina. Com o banco de dados, as informações continuam disponíveis e íntegras, mesmo que não as estejamos usando naquele momento, ou venhamos a desligar a máquina que as contém. Chamamos isto de persistência dos dados.

4- Existência de múltiplas interfaces para os usuários: os SGBD's fornecem aos seus usuários vários mecanismos de interface e acesso aos dados, que vão desde linha de comando, aplicações de acesso até aplicações web. Além das interfaces aos usuários, também é comum encontrarmos várias interfaces, para que as aplicações cheguem aos seus dados, locais ou remotos, estruturados ou semiestruturados, com drivers ou diretamente, entre outras formas.

5- Mecanismos de backup e recuperação de falhas: os mecanismos de banco de dados fornecem recursos para fazermos backup e restauração das informações, quando necessário. Existem, inclusive, SGBD's que o fazem de forma automática e transparente ao usuário. Estes mecanismos ainda são úteis para as situações, nas quais ocorreram falhas no processo e, a situação anterior precisa ser restaurada pelo próprio sistema gerenciador.

6- Representação de relacionamentos complexos entre os dados: as relações entre os diferentes dados armazenados em uma base podem ser simples ou complexas. Os bancos de dados vão mantê-las e disponibilizá-las da mesma forma para as aplicações.

7- Manutenção de restrições de integridade: nossos dados precisam de várias restrições, para serem considerados íntegros. Um SGBD deve oferecer recursos para definir e impor tais restrições, as quais vão desde os tipos de dados que cada um deve ter (inteiros, reais, caracteres, etc.), até garantir que um dado não indique algo não válido. Por exemplo, naquela situação de um dado referenciar outro com o cadastro da localização (CEP, cidade e estado), o primeiro dado não pode indicar outro que não exista ou não seja válido. O SGBD pode ser um importante recurso para dar tais garantias.

8- Adoção/imposição de padrões: quem administra uma base de dados, chamado de DBA (Database Administrator), pode estabelecer e impor determinados padrões, quanto aos dados e sua forma de tratamento, tornando as tarefas de comunicação e interoperação mais fáceis e completas.

9- Desenvolvimento das aplicações: o uso de banco de dados reduz o tempo final de desenvolvimento de aplicações, além de permitir que usemos a prática de desenvolvimento em camadas[1].

10- Flexibilidade: com o tempo, pode ser que tenhamos necessidade de trocar a

1 No desenvolvimento em camadas, separamos a implementação da regra de negócio (programa em si) da sua parte visual (interface) e do seu armazenamento de dados. Veja a secção 1.5.5, para mais detalhes desta arquitetura.

estrutura de um banco de dados, ou até mesmo todo o banco. Isto pode acontecer por problemas de escalabilidade, ou de novos recursos que são necessários aos sistemas. Os gerenciadores mais modernos permitem que certos tipos de mudança possam ser realizados, sem que afete os dados armazenados.

11- Atualidade da informação disponível de forma instantânea: como os usuários manipulam a mesma informação, muitas vezes, de maneira e com finalidade diferentes, é importante que as atualizações realizadas estejam disponíveis aos demais usuários, de forma instantânea e permanente. Para isso, existem, nos bancos de dados, sistemas de gerenciamento de concorrência (como chamamos o acesso simultâneo ao mesmo dado) e de recuperação de falhas, caso haja necessidade.

12- Economia de escala: através dos bancos de dados, podemos fazer com que aplicações diferentes usem os mesmos dados. Isto minimiza a sobreposição entre as atividades de processamento de dados, bem como a redundância deles. De uma forma simples, o uso de banco de dados torna o processamento de informação mais organizado e controlado.

Apesar dos SGBD's trazerem claras vantagens em nossas aplicações, existem casos em que podem não ser adequados ou as melhores soluções indicadas. Elmasri e Navathe (2019) citam algumas das situações, nas quais podemos desejar usar arquivos comuns, em vez das bases de dados:

1. Aplicações simples e bem definidas, onde não se espera mudanças.
2. Requisitos rigorosos de aplicações, em tempo real, que podem não ser cumpridos com as tarefas extras que o banco executa, para cada dado gravado ou manipulado.
3. Sistemas embarcados com capacidade de armazenamento restrita ou limitada.
4. Acesso de múltiplos usuários aos dados não é possível, útil ou recomendado.

1.4 USUÁRIOS DE UM SGBD

Em pequenas bases de dados de uso pessoal, uma mesma pessoa faz todas as tarefas necessárias, para definir e manter os dados. Porém, em organizações maiores, ou em situações mais complexas, muitos tipos de usuário acessam e trabalham sobre os dados armazenados. Vejamos, então, os diferentes perfis de usuários de um sistema gerenciador de banco de dados:

1. Administrador da base de dados (DBA)

Este é o profissional, responsável por: administrar todos os recursos disponíveis na base de dados; autorizar, controlar e monitorar todos os acessos ao SGBD e a base de dados; realizar ações de segurança, como backups sistemáticos e manutenções; monitorar e controlar o desempenho do sistema, podendo recomendar atualizações de hardware e software, necessárias ao bom andamento dos serviços. Em grandes organizações, talvez este seja um dos profissionais de banco de dados mais necessários e importantes.

2. Projetista de banco de dados

Os projetistas são responsáveis por identificar quais são os dados a serem armazenados, além de escolher as melhores estruturas para esta tarefa. É de sua responsabilidade a comunicação com os diversos usuários dos sistemas, para entender as suas reais necessidades de informação, devendo este profissional criar um projeto que atenda a elas. O projetista também é responsável por definir as diferentes visões que os usuários deverão ter do banco de dados e das suas informações.

3. Analistas de sistemas e programadores de aplicações

Os analistas identificam as necessidades dos usuários finais, e de que tipo de demanda precisam durante a aplicação. Depois, os programadores têm como tarefa transformar estas especificações em programas funcionais que estarão à disposição dos usuários. Os dois profissionais também necessitam ter acesso e fazer uso dos dados armazenados em um SGBD.

4. Usuários finais

São aqueles, cujo desempenho da função implica acessar e manipular as informações do banco de dados, via aplicação ou diretamente. Este tipo de usuário é aquele que apresenta uma maior variabilidade de afinidades com o sistema, indo desde usuários casuais e iniciantes, até outros permanentes e sofisticados.

1.5 HISTÓRICO

Com o surgimento dos primeiros computadores programáveis, no início da segunda metade do século passado, e com os primeiros programas elaborados pelos programadores, foi natural a necessidade de armazenar os dados nas aplicações.

Inicialmente, isso foi realizado, de forma incorporada à própria linguagem de

programação, como no caso da Linguagem Cobol, uma das primeiras linguagens de uso comercial, de tal forma que era difícil dissociarmos a linguagem de programação de como os dados eram armazenados, embora fosse separado.

Nas linguagens em que isso não era oferecido, como a Linguagem C, comum na época, cada entidade diferente de representação era um arquivo armazenado em disco. As operações de manipulação dos dados eram realizadas, através dos recursos de manipulação de arquivos oferecidos pelas linguagens e/ou sistema operacional. Com isso, não havia muitos mecanismos de controle e segurança, além do acesso ser demasiadamente lento e desorganizado.

Assim, surgiram os primeiros bancos de dados relacionais, já no final dos anos 60 e início dos 70. Foi nesta época que a IBM lançou o System R, um sistema gerenciador de banco de dados relacional, que foi o precursor do IBM Db2 atual. Também existiam alguns bancos em rede, como o CODASYL, e hierárquicos, como o do Cobol, dBase e outros. Outro banco de dados relacional, muito popular até pouco tempo atrás, o Ingres, também é originário deste período.

Um dos grandes problemas nos primeiros bancos de dados era que eles não ofereciam mecanismos suficientes para a abstração de dados e, ainda, programas e dados não eram independentes. Nestes sistemas, os relacionamentos conceituais entre os dados eram implicados e transcritos para o armazenamento e posicionamento físico dos dados no disco.

Os sistemas antigos também forneciam uma única forma de acesso aos dados, a qual era via programação, fazendo com que a implementação de novas consultas e transações fosse custosa e demorada.

Os bancos de dados relacionais iniciais, originários do trabalho de um pesquisador da IBM, chamado Edgar Codd, em um documento de título "A Relational Model of Data for Large Shared Data Banks", em 1970, se propunham a separar o armazenamento físico dos dados da sua representação conceitual, além de fornecer uma boa base matemática, para a representação de consultas aos dados, a álgebra relacional[2].

No final dos anos 80, para tornar o processo mais confuso, surgiram as primeiras linguagens de programação orientadas a objetos. Com elas, veio a necessidade de armazenar e compartilhar objetos e estruturas complexas que estavam presentes na programação. Neste período, nasceram os Bancos de Dados Orientados a Objetos (BDOO), os quais incorporavam alguns recursos dos bancos relacionais e outros do paradigma OO, como dados abstratos, encapsulamento e herança. A complexidade deste modelo e a falta de um padrão, aceito pela comunidade científica, contribuíram para o pouco uso desta abordagem.

2 Vamos falar de Álgebra Relacional no início da unidade 3 deste caderno.

Mais recentemente, os bancos de dados relacionais, que representam a abordagem mais comum nos SGBD, incorporaram aqueles conceitos do paradigma OO e tornaram-se Sistemas de Gerenciamento de Banco de Dados Objeto-Relacionais (SGBDOO).

Atualmente, estamos vivenciando o surgimento e a popularização de outro modelo de sistema de banco de dados. Os novos bancos são conhecidos pelo acrônimo NoSQL. Originalmente, a sigla era referenciada como "no SQL" (não SQL), mas, posteriormente, foi redefinido como "Not Only SQL" (não apenas SQL). Este novo sistema busca fornecer mecanismos para armazenamento e recuperação de dados, sendo modelados de formas não tabulares (tabelas), como nos bancos relacionais. Por exemplo, os bancos precisam tratar de objetos complexos, como imagens, vídeos etc., não só com o elemento, mas também com o seu conteúdo. Isto é complexo para ser realizado nos bancos relacionais.

A crescente popularização das redes sociais, bem como a geração de conteúdo, de e para dispositivos móveis, fez com que o trabalho de armazenamento de dados para uso em ferramentas analíticas (análise quantitativa dos dados) começasse a ter problemas com escalabilidade e custo de manutenção. Bases relacionais apresentam recursos de escalabilidade, mas isto é custoso e demorado, o que nem sempre é suportado pelas novas tecnologias. Já os bancos NoSQL suportam, de forma mais barata e menos trabalhosa.

Embora o crescimento dos bancos NoSQL seja notório, e se faça necessário o estudo desta arquitetura, nosso caderno não abordará o assunto. O conteúdo, após a segunda unidade, será voltado para os bancos relacionais, que representam a maioria dos bancos usados em aplicações de tratamento de sistemas de informação.

1.6 CONCEITOS E ARQUITETURA DO SGBD

Um Sistema Gerenciador de Banco de Dados tem uma arquitetura própria, de acordo com as partes e elementos que o compõem. Vamos conhecer cada uma das suas partes.

1.6.1 MODELOS DE DADOS

Quando usamos bancos de dados, estamos, implicitamente, abstraindo dados do mundo real, para representá-los no SGBD. Esta abstração de dados se refere a centrarmos a atenção em alguns detalhes dos objetos que nos interessam e relegarmos outros, que estão fora do contexto da aplicação atual. Nesse caminho, os modelos de dados oferecem

os meios práticos para alcançarmos tal abstração.

Segundo Elmasri e Navathe (2019), "um modelo de dados é uma coleção de ferramentas conceituais para a descrição dos dados, relacionamentos, semântica e restrições de consistência". É necessário entendermos que modelo de dados, esquema e instância dos dados são coisas distintas.

Enquanto o modelo representa o conjunto de conceitos usados para descrever a estrutura de um banco de dados, envolvendo a abstração necessária, bem como o tipo de dados e seus relacionamentos e operações, o **esquema** é uma descrição da estrutura de um banco de dados, de acordo com determinado modelo que foi escolhido. Por **instância** entendemos o conjunto de dados que estão armazenados no banco de dados, em um determinado instante.

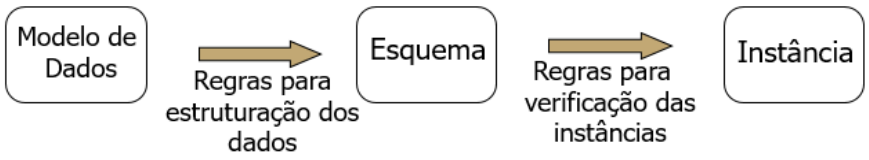


Figura 3 - Relação entre o modelo de dados, seu esquema e uma determinada instância do mesmo.
Fonte: Elaborado pelo autor (Junho/2020).

Para entendermos melhor a relação apresentada na Figura 3, considere certa aplicação de banco de dados, necessária para uma pequena empresa. Ela precisa de uma base de dados, para armazenar informações relativas aos seus clientes e aos atendimentos destes (chamamos isto de CRM - *Customer Relationship Management*). Podemos, então, definir um modelo de dados, com algumas entidades, tais como: clientes, vendas, contatos, vendedores e outras informações.

Quando analisamos cada uma das entidades, ao delinear os seus componentes, como por exemplo, definir a entidade Cliente, sendo composta de nome, endereço, telefone, e-mail e outros, estamos fornecendo um esquema de representação do modelo. Assim, os dados de um cliente específico, do João por exemplo, é uma instância.

Desta forma, a instância representa o estado do banco de dados em um determinado momento do tempo. Um banco de dados transita por vários estados (e tem instâncias distintas) durante o seu uso: ele inicia vazio, vai ganhando informações e muda com frequência. Por outro lado, o esquema do banco não muda, independente do ponto do tempo, no qual olhamos para ele. Este esquema faz parte dos metadados que estão armazenados no catálogo do SGBD.

Figura 4 - Arquitetura de três esquemas.
Fonte: Elmasri e Navathe (2019).

A forma de estruturar as diferentes visões do banco de dados, apresentada na Figura 4, foi denominada pelos autores Elmasri e Navathe (2019) como a arquitetura em três esquemas. Neste esquema, há um nível interno que deve descrever a estrutura física do armazenamento no banco de dados. No nível conceitual, o esquema descreve a estrutura do banco de dados, ocultando detalhes da implementação e visão os usuários do sistema.

Do ponto de vista conceitual, o modelo de dados possui categorias ou tipos diferentes:

1. Modelo Conceitual

É uma representação de alto nível e com foco no ponto de vista do usuário. Este é o primeiro modelo a ser desenvolvido, pois é de fácil compreensão, até mesmo pelo usuário final. Ele também não traz nenhuma limitação, ligada a tecnologias específicas da sua implementação.

Este modelo é utilizado para descrever a estrutura de um banco de dados de uma forma mais próxima da percepção dos usuários e não considera aspectos da implementação. Seus elementos são, normalmente, as entidades, seus atributos e os relacionamentos.

Os mais usuais e frequentes são: Modelo Entidade-Relacionamento (MER) e os modelos orientados a objetos, como o modelo de classes da linguagem UML.

2. Modelo Lógico

O modelo lógico agrega detalhes da implementação e trabalha com regras e limitações de recursos que serão utilizados na sua realização. Com ele, também definimos elementos (atributos) que serão as chaves de acesso para a estrutura.

Este modelo é utilizado para descrever a estrutura formal de um banco de dados, definindo como o banco será manipulado, através do SGBD. Em função disso, ele é mais dependente das estruturas físicas de armazenamento que o modelo conceitual.

Os modelos lógicos mais usuais são: o modelo relacional (que será tratado de forma aprofundada neste material), o modelo em rede e o modelo hierárquico.

SAIBA MAIS

Caso deseje maiores informações sobre os modelos em rede e hierárquico de banco de dados, que não são aprofundados no caderno, busque elementos nos livros [ELMASRI e NAVATHE, 2019], [DATE,2004] ou [HEUSER,2009].

3. Modelo Físico

Este modelo demonstra os dados sob o ponto de vista físico, levando em conta todas as regras e limitações do banco de dados. É a partir do modelo físico que o banco de dados final será montado. Este modelo caracterizará cada um dos dados a serem usados, com seus tipos e domínios de informação.

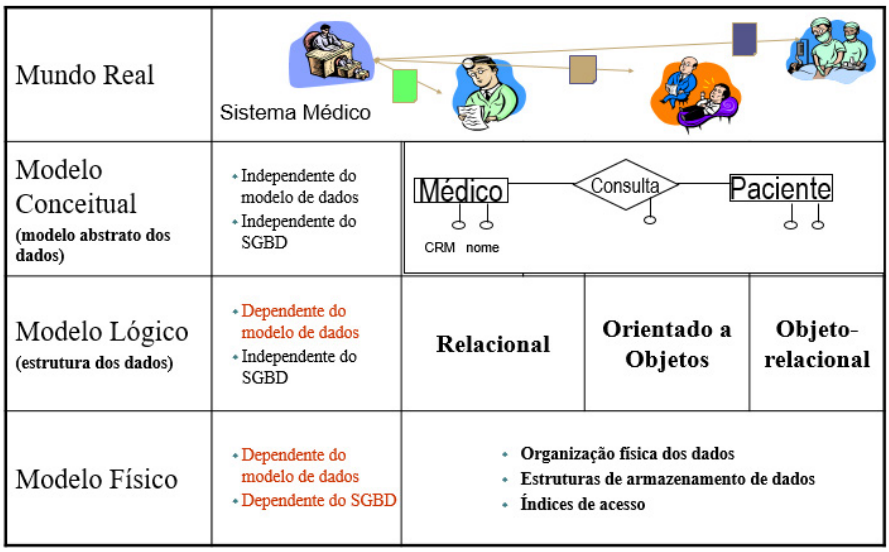


Figura 5 - Diferentes níveis do modelo de dados.
Fonte: DATE (2004).

Na Figura 5, há uma pequena estrutura que espelha todos os níveis dos modelos de dados. Nela, está representado um sistema médico, para armazenar as relações entre médicos e seus pacientes. Observe que o modelo conceitual estabelece as entidades

(médico e paciente), e uma relação entre eles, chamada de consulta. Depois, no nível lógico, faz-se necessária a definição da arquitetura do banco de dados que abrigará o modelo conceitual. Definem-se modelos, como o relacional ou orientado a objetos. Finalmente, temos o modelo físico, o qual indica como "fisicamente" os dados serão armazenados no nosso banco de dados.

1.6.2 INDEPENDÊNCIA DE DADOS

Usando a arquitetura em três camadas, conforme mencionado acima, podemos entender melhor o conceito de independência de dados, que nada mais é do que a capacidade de mudarmos o esquema presente em um dos níveis do SGBD, sem que isso altere os demais níveis.

Sob este ponto de vista, existem dois níveis distintos de independência dos dados: o nível lógico e o físico.

- **Independência lógica**

  Esta é a capacidade de alterar o modelo conceitual do banco de dados, sem ter que modificar os programas que o usam, ou qualquer outro ponto externo neste nível. Podemos alterar o esquema conceitual para expandir o banco de dados, alterar restrições ou, até mesmo, para reduzir o tamanho do banco.

- **Independência física**

  Esta é a capacidade de alterar o esquema interno do banco de dados, sem modificar o esquema conceitual. Por consequência, o nível mais externo também não precisará ser mudado. Isso pode ser necessário, porque alguns arquivos foram reorganizados, para melhorar o desempenho ou por alguma atualização.

A maioria dos SGBD's fornece independência física dos dados, pois detalhes, como a localização física dos arquivos, seus formatos e aspectos do hardware onde eles estão, não são relevantes no nível conceitual ou da aplicação.

Já a independência lógica é mais complicada de ser obtida, uma vez que as mudanças nos aspectos estruturais da base de dados são, normalmente, levadas em consideração, em algum nível, durante a implementação das aplicações.

1.6.3 DIFERENTES LINGUAGENS DO SGBD

Um SGBD precisa oferecer aos seus diferentes usuários linguagens que permitem manipular as informações armazenadas neles. Estas linguagens oferecem diferentes níveis de finalidades e acessos. Vejamos:

- **Linguagem de Definição de Dados (DDL – Data Definition Language)**

É utilizada para definir o esquema do banco de dados. O resultado do processamento desta linguagem é a criação dos elementos no banco de dados e suas respectivas informações (metadados) no catálogo.

Se usarmos a Linguagem SQL[3] como referência e exemplo, podemos listar alguns comandos, como: CREATE, para criar as tabelas no banco; ALTER, para alterar a estrutura e DROP, para remover ou apagar estas tabelas, sendo representantes da DDL nesta linguagem.

- **Linguagem de Manipulação de Dados (DML – Data Manipulation Language)**

Esta é a parte da linguagem que permite a manipulação dos dados armazenados no banco de dados. Através dela, podemos interagir com os dados armazenados, fazendo inserções, alterações e remoções das informações.

Para que os elementos da DML possam atuar, são necessárias informações que estão armazenadas no catálogo de dados, mas isto é feito de forma transparente à aplicação, ou seja, não precisamos tomar conhecimento do caminho de acesso à informação, a menos que seja necessário, por algum motivo.

Ainda usando o SQL como referência básica, os comandos INSERT, CREATE e DELETE são alguns dos representantes da DML no SGBD. O primeiro deles faz a inserção de informações, o segundo as altera e o último remove uma informação qualquer.

- **Linguagem de Consulta de Dados (DQL – Data Query Language)**

Talvez seja a operação mais realizada em um SGBD, na maioria das aplicações, e, por conta disso, é uma das mais privilegiadas, em termos de recursos de apoio que a tornam mais eficiente. Por definição, então, a DQL fornece os recursos necessários, para que possamos consultar os dados armazenados.

Por outro lado, de forma curiosa, a Linguagem SQL apresenta um único representante desta linguagem, que é o comando SELECT. Embora seja único, ele é um dos comandos com mais variação nos formatos de uso.

3 A Linguagem SQL é objeto de estudo na Unidade 3 deste caderno.

- **Linguagem de Transação de Dados (DTL – Data Transaction Language)**

Como nossas bases de dados são de uso múltiplo, isto é, vários usuários podem usar as mesmas informações e, ao mesmo tempo, faz-se necessária a existência de uma linguagem que permita controlar as transações de acesso aos dados.

Uma transação é uma unidade de trabalho, executada dentro de um sistema de gerenciamento de banco de dados, sobre um banco de dados, sendo tratada de maneira coerente e confiável, independente de outras transações. Uma transação, geralmente, representa qualquer alteração em um banco de dados.

Usando mais uma vez o SQL, a DTL está representada nos comandos BEGIN TRANSACTION, COMMIT e ROLLBACK, os quais servem para abrir uma transação "manualmente" no banco, que será encerrada com sucesso (COMMIT) ou desfeita (ROLLBACK) no final.

- **Linguagem de Controle de Dados (DCL – Data Control Language)**

Finalmente, existe a DCL, que é a linguagem responsável pelo controle da segurança no acesso ao banco de dados. No SQL encontramos comandos como GRANT, REVOKE e DENY, os quais gerenciam a forma como os usuários acessam o banco de dados.

Nos SGBD atuais, a diferença entre estas linguagens é transparente e não chegam a ser consideradas linguagens distintas, em função da Linguagem SQL, que acabou por fornecer uma forma uniforme e semelhante para acesso a cada um dos recursos acima.

1.6.4 COMPONENTES DE UM SGBD

A Figura 6 abaixo apresenta, de forma sintética, como é a estrutura de componentes de um Sistema Gerenciador de Banco de Dados. Na parte superior dela, vemos os recursos e componentes que são mais orientados aos diferentes usuários de um SGBD. Na parte inferior, estão presentes os componentes responsáveis pelo armazenamento dos dados e da execução das transações com estes dados.

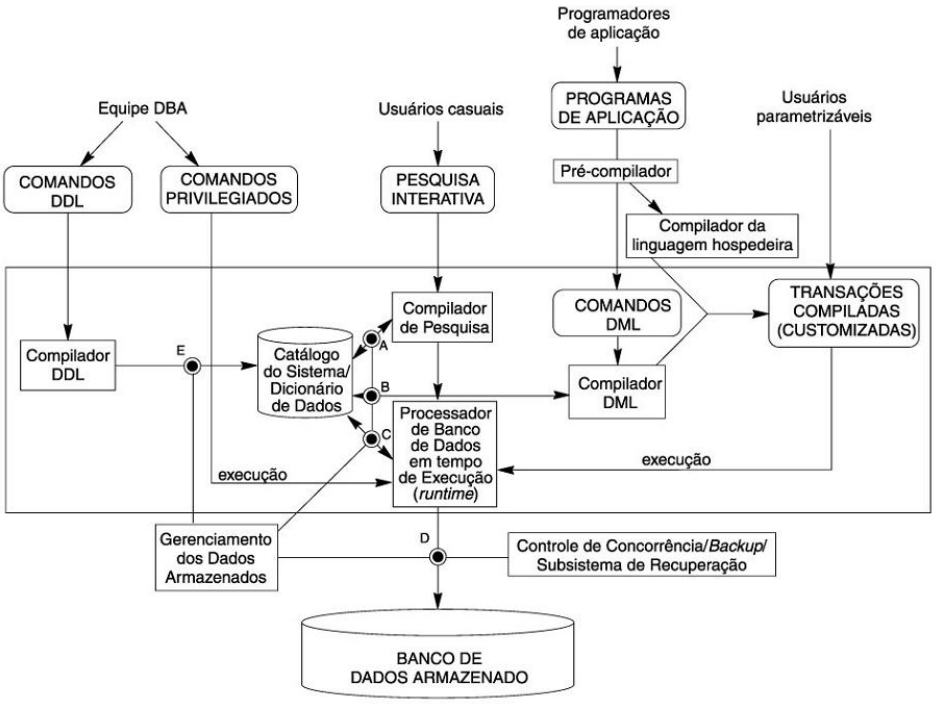


Figura 6 - Componentes de um SGBD.
Fonte Elmasri e Navathe (2019).

Pode-se observar, na parte superior da figura, as interfaces que permitem aos usuários realizarem as consultas e a manipulação de dados. Ali, aparecem claramente os usuários que administram o SGBD, que são os DBA's, para os quais são fornecidos os recursos da DDL, inclusive com alguns comandos privilegiados de acesso aos dados. Em paralelo a isso, os programas de aplicação e usuários outros têm acesso aos recursos da DML e DQL, para que possam fazer as suas transações na base.

Quando um comando da DML, DDL ou DQL é apresentado ao SGBD, ele faz uso de um compilador que traduzirá os comandos, de acordo com as necessidades do sistema. Para compilar, o SGBD faz uso de informações presentes no catálogo do banco de dados onde estão os metadados. Estando este processo pronto, o sistema repassa o comando ao nível inferior, a fim de que ele execute a ação desejada e produza a resposta, a qual será novamente enviada aos níveis superiores.

A parte inferior (Figura 6) realiza a chamada "independência física de dados", fazendo com que os níveis superiores desconheçam elementos físicos de armazenamento. Normalmente, suas funções são executadas, levando em consideração elementos relativos a hardware e software (sistema operacional), onde a base de dados está armazenada.

1.6.5 ARQUITETURAS EM UM SGBD

Um SGBD apresenta diferentes arquiteturas que são muito semelhantes às arquiteturas dos sistemas de computação em geral. São elas: centralizada, cliente-servidor e em três camadas, principalmente na web.

1. Centralizada

Em uma arquitetura centralizada, todo o processamento é realizado dentro de um computador central. Nesta modalidade, todos os recursos do SGBD, incluindo suas funções, execução dos programas de aplicação e o processamento da interface estão localizados na máquina central.

Esta arquitetura não é muito usual hoje, uma vez que os recursos de processamento estão melhores distribuídos e podemos destinar partes do funcionamento para cada um deles, sem que isso comprometa o funcionamento ou a eficiência.

2. Cliente-servidor

De forma mais frequente e usual, encontramos a arquitetura cliente-servidor para os sistemas de banco de dados, a qual usa servidores especializados, de acordo com funções específicas. Dentre eles, encontraremos servidores de arquivos, de impressão e, de forma especial aqui, o servidor de banco de dados.

Neste servidor, estará instalado o SGBD de forma integral, enquanto as aplicações e recursos dos usuários estarão armazenados em suas próprias máquinas. Para que isso funcione, é necessário que a estrutura de rede de comunicação forneça comunicação entre as máquinas dos clientes e os servidores.

Na arquitetura cliente-servidor, as máquinas dos clientes devem fornecer as interfaces necessárias, para que este faça uso dos recursos do banco de dados, além de alguns recursos de processamento local.

Uma possível variação para ela é o sistema em duas camadas. Nesta variação, os recursos de consulta e controle de transações permanecem no lado do servidor, caracterizando um servidor de consulta ou servidor de transações. Já os programas de aplicação e as interfaces são colocados no lado do cliente. Ou seja, o cliente é responsável pelo processamento dos dados e, quando necessita deles, solicita ao servidor que os busque e os envie, através da rede. Uma vez realizado o processamento, caso haja necessidade, o resultado é enviado de volta ao servidor para armazenamento ou atualização. Observe na Figura 7.

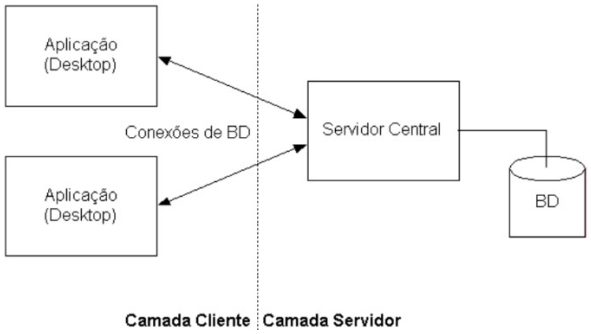


Figura 7 - Arquitetura cliente-servidor em duas camadas.
Fonte: Elaborado pelo autor (Junho de 2020).

3. Três camadas

No sistema em três camadas, comumente presentes em aplicações web, há uma separação física entre os níveis da aplicação. Esta não deixa de ser uma arquitetura do tipo cliente-servidor, mas tem características próprias.

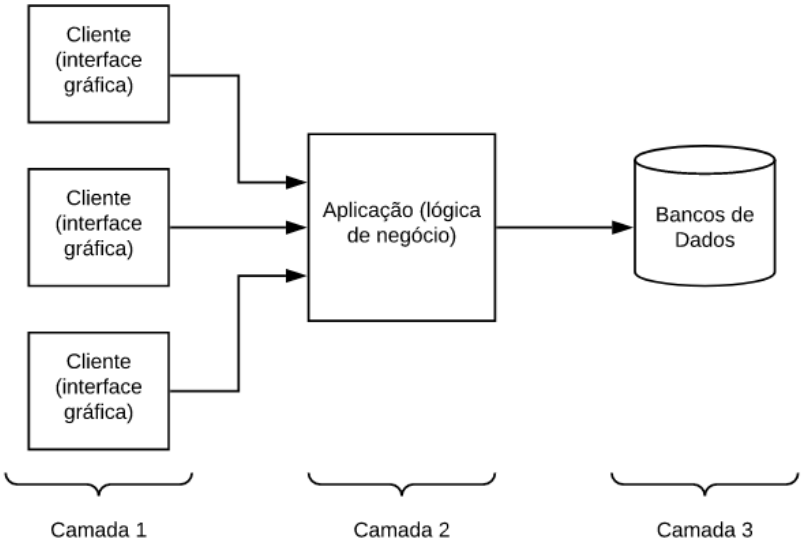


Figura 8 - Arquitetura em três camadas.
Fonte: Elaborado pelo autor (Junho / 2020).

No desenvolvimento em três camadas, vamos separar a interface do usuário, onde eles interagem com os dados, através de ações, como visualização e entrada de informações, do local em que elas foram processadas, e está localizada a chamada “Regra do Negócio”. Estas são as definições de como a empresa faz o seu negócio, isto é, define como os recursos e informações devem ser processados, de acordo com as suas políticas. Na última das camadas, está localizado o servidor de banco de dados, ou o SGBD propriamente dito.

Todo o acesso a ele se dá, por meio das redes de comunicação.

As aplicações web, conforme foi mencionado, são excelentes representantes deste tipo de arquitetura. Nela, o SGBD está localizado em uma máquina remota e acessada pela internet. A aplicação, desenvolvida em linguagens como PHP, por exemplo, roda em um servidor de aplicação, também separado dos demais, enquanto o usuário interage com estes recursos (interface), através dos navegadores e aplicativos próprios, mais uma vez fisicamente separados.

1.6.6 CLASSIFICAÇÃO DOS SGBD´S

Os sistemas gerenciadores de banco de dados podem ser classificados em três grupos distintos, quanto ao modelo de dados (conceitual) adotado, aos usuários suportados e quanto à localização dos dados. Vamos a eles:

Quanto ao modelo de dados adotado:

= > Relacionais: bancos de dados relacionais modelam os dados enquanto tabelas, organizadas em colunas, e armazena as relações existentes entre as linhas destas tabelas.

= > Hierárquico: este modelo organiza os dados na forma de uma árvore (invertida), de cima para baixo, onde dados mais acima têm hierarquia superior aos demais, semelhantes a uma estrutura de pastas no sistema operacional. As informações são organizadas em registros conectados por meio de ligações. Cada registro é uma organização de campos e cada campo contém informações.

= > Em rede: este modelo surgiu como uma extensão do modelo hierárquico, eliminando o conceito de hierarquia, permitindo que um mesmo registro estivesse envolvido em várias associações. No modelo em rede, os registros são organizados na forma de grafos, aparecendo um único tipo de associação que define uma relação entre dois tipos de registros: proprietário e membro.

= > Orientado a Objetos: é um banco em que cada informação é armazenada na forma de objetos, apenas podendo ser manipulada através de métodos definidos pela classe, na qual esteja o objeto. O conceito de banco de dados orientado a objetos é o mesmo das linguagens orientadas a objetos, havendo uma pequena diferença: a persistência de dados.

= > Objeto Relacional: é a junção do modelo relacional com o modelo orientado a objetos. Eles estendem os conceitos básicos do banco relacional, para incorporar o suporte para o modelo de dados relacional-objeto, gerenciam transações, processamento

e otimização de consultas. Este modelo passou a ter identidade de objeto, encapsulamento de operações, além de terem sido adicionados o mecanismo de herança e o polimorfismo.

= > NoSQL: busca fornecer mecanismos, para armazenamento e recuperação de dados, que são modelados de formas não tabulares (tabelas), como nos bancos relacionais. Os bancos de dados NoSQL usam uma variedade de modelos de dados para acessar e gerenciar os dados. Estes tipos de banco de dados são otimizados, especificamente, para aplicativos que exigem modelos de grande volume de dados, baixa latência e flexibilidade, cujos requisitos são atendidos, mediante o relaxamento de algumas restrições de consistência de dados dos outros bancos.

Quanto ao número de usuários:

= > Monousuários: este é um sistema que pode ser utilizado por um único usuário, durante um determinado tempo. Os bancos de dados atuais não procedem desta forma. Isso era mais frequente nos primeiros bancos de dados.

= > Multiusuários: este sistema, usual nos bancos de dados atuais, permite que múltiplos usuários façam uso simultâneo dos recursos e informações mantidos por eles.

Quanto à localização dos dados:

= > Centralizados: bancos de dados centralizados mantém todos os seus dados armazenados em um único lugar. (Atenção: não é a arquitetura centralizada apresentada antes). Neste caso, os dados ficam todos agrupados em um único lugar, o que é muito usual nas aplicações atuais.

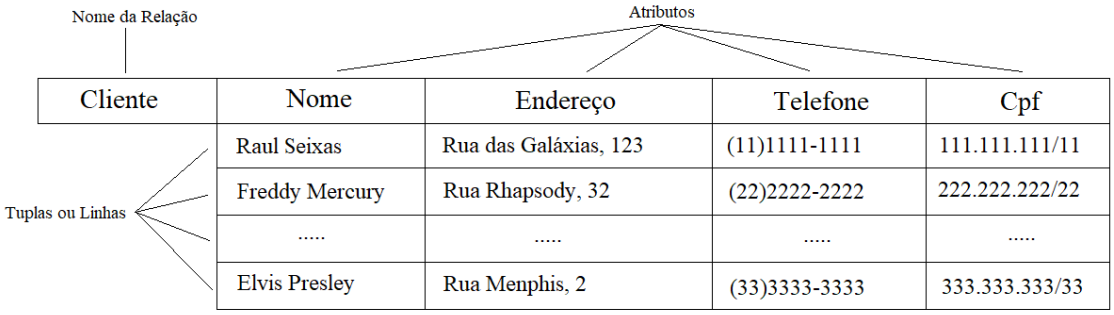
= > Distribuídos: outra forma de manter os dados do SGBD é de forma distribuída, ou seja, os dados estão separados fisicamente entre si. Isso pode ser útil por questões de segurança e performance. Por exemplo, uma empresa com várias filiais pode ter os seus dados distribuídos entre as filiais, de forma a permitir acesso mais rápido a suas informações. Por outro lado, quando a empresa quer "ver" o sistema de banco de dados, ela o enxerga como se fosse uma base única.

1.7 MODELO RELACIONAL

O modelo relacional para bancos de dados foi proposto por Edgar Codd, nos anos 70, mas somente foi aplicado nos SGBD's no início dos anos 80, ambos do século passado. A principal ideia deste modelo se baseia no conceito de que as informações da base de dados podem ser consideradas como relações matemáticas (baseadas na álgebra relacional e

teoria dos conjuntos), podendo ser representadas, de forma uniforme, através do uso de tabelas, nas quais as linhas representam as diferentes instâncias de uma entidade, e as colunas representam os seus atributos.

Desta forma, o modelo relacional representa um banco de dados como um conjunto de relações. Cada uma das relações é indicada logicamente como uma tabela de valores, em que cada uma das linhas, chamadas de tuplas, evidencia uma coleção de dados relacionados entre si. As linhas representam um elemento de uma entidade ou um relacionamento do mundo real. Observe isso na Figura 9.



Cliente	Nome	Endereço	Telefone	Cpf
	Raul Seixas	Rua das Galáxias, 123	(11)1111-1111	111.111.111/11
	Freddy Mercury	Rua Rhapsody, 32	(22)2222-2222	222.222.222/22
	.....	.....	.....	.....
	Elvis Presley	Rua Menphis, 2	(33)3333-3333	333.333.333/33

Figura 9 - Modelo relacional.
Fonte: Elaborado pelo autor (Junho de 2020).

Ainda usando a Figura 9 como referência, podemos visualizar alguns conceitos básicos deste modelo. No modelo relacional, cada uma das linhas da relação (tabela) são chamadas de **tuplas**. Cada uma das colunas da tabela recebe um nome, que chamamos de **atributo**. Na Figura 8, por exemplo, na qual a relação se chama Cliente, observamos os atributos: Nome, Endereço, Telefone e CPF. Ainda na tabela, cada uma das linhas contém o conjunto completo de uma representação do mundo real, composta dos seus valores para cada atributo.

Quando definimos um atributo para uma tabela, precisamos sempre levar em conta o seu domínio, ou seja, qual é o conjunto de valores que pode aparecer naquela coluna. Por exemplo, se possuímos um atributo chamado idade, ele terá como domínio um número inteiro positivo, já que não existem idades negativas. Se possuímos o salário de uma pessoa como atributo, o domínio será composto de números reais, e assim por diante, para cada um dos atributos.

1.7.1 CARACTERÍSTICAS DO MODELO RELACIONAL

Segundo Date (2004), ao definir o modelo relacional, Codd estabeleceu um conjunto de regras que devem ser usadas para verificar a fidelidade de um banco de dados ao modelo relacional. Por mais que poucos bancos consigam atender a todas elas, este conjunto de regras, acaba estabelecendo um conjunto desejável de características a serem buscadas nos bancos de dados. São elas:

a. Toda informação, em um banco de dados relacional, é apresentada em nível lógico, na forma de tabelas ou relações;
b. Todo dado, em um banco de dados relacional, tem a garantia de ser logicamente acessível, recorrendo-se a uma combinação do nome da tabela, um valor de chave e o nome da coluna. Isso permite acessarmos cada parte da tabela, individualmente, e de forma independente das demais informações;
c. Tratamento sistemático de valores nulos, ou para a ausência de informação, de forma a evitar erros por ausência de informação;
d. O dicionário de dados, catálogo, do banco de dados é baseado no modelo relacional. Assim, há uniformidade na forma de representação e tratamento, inclusive dos metadados;
e. Há uma linguagem não procedural para definição, manipulação e controle dos dados (aqui entra a Linguagem SQL);
f. Tratamento das atualizações de visões dos dados. Visões são formas dinâmicas, construídas para que os dados de uma ou várias relações possam ser visualizadas de alguma forma específica[4];
g. Tratamento de alto nível para inserção, atualização e eliminação de dados;
h. Independência física dos dados. Mudanças na memória e no método de acesso; criação de um novo índice ou criação de uma nova coluna não podem influenciar o método de utilização dos dados pelos usuários finais;
i. Independência lógica dos dados, ou seja, mudanças nas colunas, como tamanho por exemplo, não afetam as demais camadas da aplicação;
j. Restrição de Integridade (Identidade, Referencial e Domínio). Estas são restrições que devem ser aplicadas a cada uma das relações, de forma a tornar a representação e o uso mais adequados ao modelo[5];
k. Independência de Distribuição dos dados;
l. Não subversão das regras de integridade ou restrições quando se usa uma

4 Veja a Unidade 3, para maiores detalhes sobre as visões.
5 Veja a Unidade 2, para maiores detalhes sobre as restrições de integridade.

linguagem hospedeira.

Com a presença das características (ou boa parte delas), um banco de dados relacional, ainda baseado nas definições de Codd, apresentarão as seguintes vantagens:

- Independência total dos dados;
- Permite múltiplas visões dos mesmos dados;
- Redução do trabalho necessário ao desenvolvimento de aplicações;
- Maior segurança no acesso aos dados;
- Maior agilidade para consulta e atualização das informações.
- Qualidade dos dados garantida por restrições de integridade da base.

1.7.2 DEFINIÇÃO DA RELAÇÃO

Em uma base de dados relacional, cada uma das relações presentes é definida por um esquema básico de representação. Definimos um esquema da seguinte forma:

$$R(A_1,A_2, ...,A_n)$$

Neste esquema, **R** representa o nome da relação e $\mathbf{A}_i$ é o atributo da posição **i**. Cada atributo $\mathbf{A}_i$ é o nome de um papel desempenhado por algum domínio **D** no esquema da relação **R**.

Vejamos alguns exemplos de relações definidas dentro do modelo (a primeira delas é relativa ao exemplo da Figura 8):

Cliente (Nome, Endereço, Telefone, CPF).
Fornecedor (Nome, Endereço, CEP, Telefone, CNPJ).
Produto (Código, Nome, Fabricante, Preço).

Nesse contexto, podemos também definir que a relação **R** será formada por um conjunto de tuplas:

$$R = \{t_1, t_2, t_3, ..., t_m\}$$

Cada uma das tuplas $\mathbf{t}_i$ é composta de uma lista ordenada de valores:

$$t_i = < v_1, v_2, v_3, ..., v_n >$$

É fundamental perceber que, em cada uma das tuplas, os valores são ordenados, de acordo com a declaração dos atributos da relação. Do ponto de vista do seu uso, esta ordenação não é relevante, exceto para determinados tipos de dados de tamanho variável.

Por outro lado, dentro da relação, cada uma de suas tuplas não é ordenada. Assim, dentro da relação ou tabela, as informações são ordenadas e armazenadas, conforme a sua inserção, ou seja, tuplas inseridas primeiro, virão à frente das demais tuplas da relação.

Ainda, quando verificamos uma relação **R** qualquer, percebemos que o conjunto de dados, o qual forma uma determinada tupla $\mathbf{t}_i$, é sempre atômico e, preferivelmente, único. Atributos compostos ou multivalorados não são permitidos, uma das características a serem exploradas nas restrições de integridade da próxima unidade.

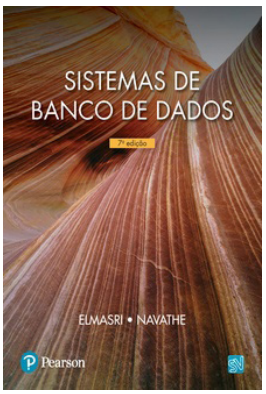
Além disso, uma tupla não pode ser totalmente nula, embora algum atributo específico possa ser usado para representar algo que não é conhecido para aquele domínio. Por exemplo, caso tenhamos um atributo que deva armazenar uma data, e ela for informada de maneira errada ou inválida, seu atributo poderá conter um valor nulo (*null*, na representação forma do atributo).

SAIBA MAIS

Saiba mais a respeito dos conceitos básicos e introdutórios sobre Bancos de Dados e o modelo relacional nos vídeos (recomendo os 3 primeiros da série, neste momento), acessíveis pelo endereço: https://www.youtube.com/watch? v=Q_KTYFgvu1s&list=PLucm8g_ezqNoNHU8tjVeHmRGBFnjDIlxD.

SUGESTÃO DE LIVRO

O livro Sistema de Banco de Dados, de autoria de Ramez Elmasri e Shamkant Navathe é uma das principais referências na área de banco de dados. Todos os conceitos apresentados nesta unidade (e nas demais) estão presentes nele, de forma clara e didática. Vale a leitura!



CONSIDERAÇÕES FINAIS

Nesta primeira unidade, foram apresentados os conceitos iniciais dos Sistemas Gerenciadores de Banco de Dados – SGBD's, além de como eles são organizados. Estes sistemas são, na verdade, um grande conjunto de softwares, cuja finalidade específica é gerenciar o armazenamento e uso de informações em bases de dados.

Quando falamos de banco de dados, estamos simplificando a definição e estamos nos referindo ao conjunto de ferramentas que compõem o SGBD, inclusive ao próprio banco, o qual é responsável pelo armazenamento dos dados.

Existem vários modelos diferentes de bancos de dados, alguns mais antigos, como os modelos de rede e hierárquicos, enquanto outros são mais atuais, como o modelo objeto-relacional. Permeando a todos, um modelo permanece importante para o projeto e desenvolvimento de aplicações, o modelo relacional, no qual um grande conjunto de SGBD's atuais é baseado. Assim, este caderno dedicará grande parte do seu conteúdo para tratar deste modelo.

EXERCÍCIO FINAL

1.Sobre definições de banco de dados, analise as assertivas e assinale a alternativa que aponta as corretas.

I. Um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito.

II. Um banco de dados representa alguns aspectos do mundo real, sendo chamado, às vezes, de minimundo ou de universo de discurso (UoD).

III. Um banco de dados pode ser uma coleção lógica e coerente de dados com algum significado inerente.

a) Apenas I e II.

b) Apenas I e III.

c) Apenas II e III.

d) Todas estão incorretas.

e) Todas estão corretas.

2. O conceito "dado que é associado a cada ocorrência de uma relação ou entidade", refere-se a:

a) Atributo
b) Relação
c) Domínio
d) Tupla
e) Modelo de dados

3. Em um modelo de dados relacional, o tipo de dado que descreve os tipos de valores, os quais podem aparecer em cada coluna, é chamado de:
a) Atributo
b) Relação
c) Domínio
d) Tupla
e) Modelo de dados

4. "Representa o banco de dados como uma coleção de relações. Informalmente, cada relação é semelhante a uma tabela de valores ou, até certo ponto, a um arquivo plano de registros. Ele é chamado de arquivo plano, porque cada registro tem uma simples estrutura linear ou plana". A descrição acima está intimamente ligada ao conceito do:
a) Sistema Gerenciador de Banco de Dados.
b) Modelo de Dados Relacional.
c) Banco de Dados em Rede.
d) Independência do Modelo Lógico de Dados.
e) Tupla ou Linha da Relação.

5. Uma característica fundamental da abordagem de um banco de dados é que seu sistema possui não apenas o banco de dados, mas também uma completa definição ou descrição da estrutura do banco e suas restrições. Esta definição fica armazenada em um local que contém informações, como a estrutura de cada arquivo, o tipo e o formato de armazenamento de cada item de dado, e várias restrições sobre os dados. A informação armazenada neste local tem uma certa denominação e descreve a estrutura do banco de dados primário. O local, ao qual o texto se refere, e a denominação da informação nele armazenada são, correta e respectivamente:
a) Modelo de dados – tupla
b) Modelo de dados – transação
c) SGBD – transação
d) Catálogo – metadados
e) Memória - atributo

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados**. 1. ed. São Paulo: Érica, 2014.

AMADEU, Cláudia (org). **Banco de Dados**. São Paulo: Pearson, 2014.

BARBOZA, Fabrício. **Modelagem e Desenvolvimento de Banco de Dados**. Porto Alegre: SAGAH, 2018.

CARDOSO, Virginia. CARDOSO, Giselle. **Sistemas de Banco de Dados: uma abordagem introdutória e prática**. São Paulo: Saraiva, 2012.

CARDOSO, Virginia; CARDOSO, Giselle. **Linguagem SQL**: **fundamentos e práticas**. São Paulo: Saraiva, 2013.

DATE, Chris. Introdução a Sistemas de Banco de Dados. 8.ed. São Paulo, Campus: 2004.

ELSMARI, Ramez. NAVATHE, Shamkant**. Sistemas de Banco de Dados**. 7. ed. São Paulo: Pearson, 2019.

HEUSER, Carlos. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

MACHADO, Felipe. Banco de Dados: Projeto e Implementação. 4. ed. São Paulo: Érica, 2014.

MEDEIROS, Luciano. **Banco de dados: princípios e práticas**. Curitiba: Intersaberes, 2013.

MANNINO, Michael V. **Projeto, desenvolvimento de aplicações e administração de banco de dados**. 3. ed. Porto Alegre: AMGH, 2014.

RAMARKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados**. 3. ed. Porto Alegre: AMGH, 2011.

TEOREY, Toby et. al. **Projeto e Modelagem de Banco de Dados**. 2. ed. São Paulo: Elsevier, 2014.

unidade 2

PROJETO DE BANCO DE DADOS

2.1 INTRODUÇÃO À UNIDADE

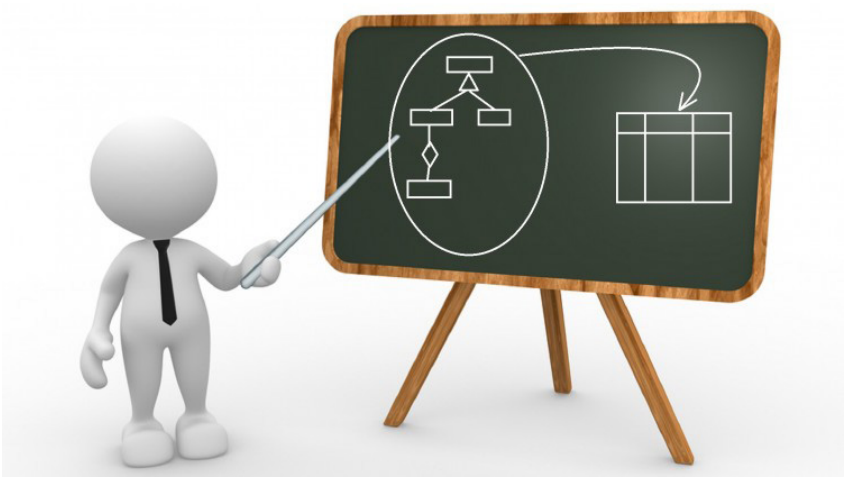


Figura 10 - Projeto de Banco de Dados.
Fonte: <https://www.udemy.com/course/modelagem-de-dados-em-banco-de-dados-relacional/>
Acesso em: Junho de 2020.

Quando necessitamos de um banco de dados para aplicações reais, o primeiro passo necessário é fazermos o projeto do banco de dados. Projetar um banco de dados é, na verdade, pensar cada pedaço deste banco, em detalhes, antes de transformá-lo em uma aplicação real.

O projeto do banco é um processo, realizado em várias etapas, que culmina com o banco de dados efetivamente disponibilizado para alguma aplicação fazer uso. Este processo é fundamental para a eficácia e a eficiência da representação, ou seja, para o seu sucesso.

Nesta unidade, entenderemos os primeiros passos. Em primeiro lugar, analisaremos quais são as restrições de integridade que uma base de dados deve respeitar. Com elas, podemos garantir que a representação permaneça íntegra durante toda a sua vida útil.

Após isso, estudaremos as dependências funcionais, as quais representam as possíveis relações distintas entre os diferentes atributos que foram levantados para a relação.

Algumas das dependências funcionais implicam ações que buscam otimizar a representação, evitar redundância e garantir a busca da eficiência na representação. Neste ponto, entram as formas normais e o processo de obtê-las, que se chama normalização, itens que compõem as duas partes finais desta unidade.

Os principais objetivos de aprendizagem da segunda unidade são: conhecer e saber a finalidade dos conceitos principais, utilizados na definição de bases relacionais; conhecer o modelo relacional de banco de dados e aplicar seus conceitos na construção de bases de dados; compreender e aplicar os processos de projeto de uma base de dados; conhecer as formas normais, sua aplicação em uma base de dados e a implicação de cada uma delas no projeto do banco de dados.

2.2 INTRODUÇÃO

Um projeto de banco de dados deve ser caracterizado como um processo, cujo objetivo é garantir que todas as necessidades dos usuários finais, ou as aplicações, estejam plenamente atingidas, e que as informações necessárias, para isto acontecer, tenham sido corretamente representadas e armazenadas.

Este processo tem como ponto de partida a análise de requisitos realizada durante o projeto do software, em que são estabelecidas as necessidades funcionais e não funcionais da ferramenta em desenvolvimento, impactando na disponibilidade, desempenho e confiabilidade das informações e, portanto, no banco de dados que as armazena.

O projeto de um banco de dados passa por três fases distintas: a modelagem conceitual, a modelagem lógica e a modelagem física. Cada uma destas fases pode ser assim descrita:

a) Modelagem Conceitual

Esta etapa representa os conceitos do negócio, e as associações existentes entre cada um dos conceitos. Nela, também serão representados alguns atributos que fazem parte da regra e que regulam as associações existentes. Como visto na primeira unidade, o modelo conceitual é completamente independente das tecnologias de implementação que serão utilizadas. Em função disso, não existe a necessidade de conhecimentos técnicos profundos, o que facilita o envolvimento do usuário final no processo.

Os principais produtos desta etapa são a primeira lista das relações (tabelas) e seus atributos; a lista das restrições de integridade e o modelo entidade-relacionamento que a representa[6].

b. Modelagem Lógica

Este modelo busca mostrar as estruturas de dados que devem ser implementadas, para representar a informação, bem como a característica de cada uma das estruturas, mas sempre levando em consideração o modelo de dados usado na representação do banco.

É derivada da modelagem conceitual, devendo apresentar a forma de implementação de cada uma das relações, definir quais são as suas chaves primárias e estrangeiras, garantir a presença da normalização da base, a adequação da nomenclatura dos atributos e relações que foram adotadas e, finalmente, definir quais serão os atributos a serem armazenados.

6 O Modelo ER está presente a Unidade 3 deste material.

Seu principal produto é o modelo relacional a ser adotado. Para que o projeto chegue a este modelo, existem ferramentas, como o modelo Entidade-Relacionamento, o modelo Orientado a Objetos, diagramas UML e outros. Eles serão abordados na próxima unidade.

c. Modelagem Física

Finamente, a modelagem física define a representação a ser dada do modelo lógico, enquanto implementação, em algum modelo particular de tecnologia de banco de dados. Esta etapa é derivada da modelagem lógica, podendo variar, conforme a tecnologia escolhida para a implementação do banco. Nela, o modelo lógico será traduzido em tabelas físicas do banco de dados e em restrições para o seu uso.

Nesta unidade, abordaremos conceitos que fazem parte, principalmente, da modelagem conceitual, mas também alguns aspectos da modelagem lógica da base de dados (que estará presente na Unidade 3). Aqui, compreenderemos aspectos relacionados às restrições a que um projeto deve obedecer, e como será o processo, o qual dará organização a isto.

2.3 CONCEITOS BÁSICOS

Quando é realizada a análise de requisitos, determinamos um conjunto básico de elementos a serem armazenados nas entidades do banco de dados. Por exemplo, ao verificarmos as necessidades de um pequeno sistema para gerenciamento de uma biblioteca, podemos determinar que há a necessidade de armazenarmos dados dos livros (quais?), dos autores (quais?) e dos usuários da biblioteca (quais?), para sermos minimalistas.

Tais necessidades devem ser transformadas em relações e relacionamentos, identificadas as chaves de acesso, montado um esquema inicial e definido o domínio de cada um dos atributos. Vejamos estes conceitos:

a) Entidades ou Relações

Como visto na definição dos bancos relacionais, relações ou entidades são as diferentes tabelas de valores que serão tratadas e armazenadas em nossa base de dados, as quais são organizadas em tuplas (linhas da tabela), sendo cada tupla separada por atributos,

que são os campos.
Por exemplo, na situação descrita acima, poderíamos ter, de forma inicial, as tabelas: Livro, Autor e Usuário. Em uma análise mais profunda, poderemos perceber a necessidade de termos relações, também, para as Editoras, Empréstimos e outros, que podem aparecer durante a fase de normalização.
Tomando uma das tabelas, o Autor, por exemplo, ela deve conter atributos, como Nome, E-mail e outros que sejam necessários, para o correto armazenamento das informações.

b) Relacionamentos

Voltando ao exemplo acima, como vamos saber que certo Livro foi escrito por um determinado autor (ou por vários)? Bem, uma entidade não existe de forma isolada, e sim se relacionando com outras entidades, de forma a permitir que o negócio seja implementado corretamente.
Um relacionamento é uma associação[7] entre duas entidades, a qual existe, para que possamos representar mais corretamente o mundo real. Através de um relacionamento, podemos identificar que o Livro X foi escrito pelo Autor Y. Para tanto, determinados atributos das relações devem ser identificados como elementos que podem ser usados, a fim de que a associação seja clara e univocamente identificada. Estes são os atributos, chamados de chaves, descritos a seguir.

c) Tipos de Chave

O conceito de chave é importante na modelagem de dados, pois implementa restrições que garantem a integridade referencial dos dados no banco de dados. Por chaves, entendemos um atributo (ou coluna da tabela), ou um conjunto de atributos que podem ser selecionados, para identificar, de forma única, certa tupla da entidade. Por exemplo, na entidade Livro, podemos determinar que o atributo ISBN será usado para identificar cada livro, uma vez que ele é único e individual.
Existem diferentes tipos de chaves em um banco de dados. Vejamos cada uma delas:

- Superchave: é um conjunto de um ou mais atributos que, tomados de forma coletiva, permitem identificar, de maneira única, uma entidade em um conjunto de entidades. Em outras palavras, não podem existir duas ou mais linhas da tabela com os mesmos valores de uma superchave. Acima, citamos uma superchave como exemplo, que é o ISBN do Livro. No mesmo contexto, o nome do livro não pode ser uma superchave, pois ele pode ser duplicado em livros diferentes.

7 Estas são as associações presentes no processo de modelagem, como na Linguagem UML, por exemplo.

- Chave candidata: são as superchaves de tamanho mínimo, isto é, são aquelas chaves candidatas a serem as chaves primárias de uma relação. Como superchave, elas também servem para identificar, de forma inequívoca, qualquer tupla de uma relação, porém elas não podem ser reduzidas, sem que percam a qualidade da informação.
- Chave primária (PK): depois de verificarmos todas as chaves candidatas de uma relação, determinamos uma delas, que será usada na implementação do banco de dados. Esta será a Chave Primária ou PK (*primary key*) da nossa relação, a qual sempre será única, e a menor possível, dentro da relação.
- Chave estrangeira (FK): São atributos de uma relação, os quais fazem referência a uma chave primária de outra relação (ou até, da própria). As Chaves Estrangeiras ou FK (*foreign key*) servem, então, para apontar a elementos de um relacionamento. Por exemplo, quando, dentro da entidade Livro, armazenamos o código da sua editora, estamos usando uma FK, visto que este atributo indica uma chave primária de uma relação externa àquele que estamos verificando.

d) Atributos

Uma vez que tenhamos definidas quais são as entidades que armazenaremos em nosso banco, necessitamos definir quais serão os atributos que compõem cada uma delas.

Quando observamos os atributos que foram modelados, perceberemos que existem tipos distintos deles. Os principais são:

- Compostos: os atributos compostos podem ser divididos em partes menores, as quais seriam atributos básicos simples, mas com significado independente. Um exemplo, seria um atributo chamado endereço, que poderia ser dividido em atributos mais simples, tais como rua, cidade, estado e CEP.
- Simples: são atributos atômicos, que não podem ser subdivididos. Um exemplo seria o CEP de um endereço.
- Monovalorados: são atributos que possuem um único valor para um elemento em particular da relação. Por exemplo, a idade de uma pessoa é um atributo deste tipo.
- Multivalorados: são atributos que possuem um ou mais valores para um mesmo elemento de uma relação. Por exemplo, poderíamos ter o atributo telefone em uma relação. Uma mesma pessoa pode ter vários números de telefones.
- Derivados: são atributos que possuem alguma relação entre si. Por exemplo, os atributos idade e a data de nascimento de uma mesma pessoa, pois podemos determinar a sua idade, a partir da data de nascimento. Assim, idade é derivado

de data de nascimento. A derivação pode ser indireta também. Por exemplo, em uma relação, podemos armazenar a quantidade de empregados que certo departamento tenha, o que poderia, também, ser descoberto, a partir de uma contagem realizada no cadastro de funcionários deste departamento.

- Nulos: são atributos que não necessitam obrigatoriamente de um valor, para serem armazenados na relação. Por exemplo, podemos ter o número do telefone da pessoa em nossa relação, mas este campo será nulo, caso a pessoa em questão não possua um telefone.

e) Domínio dos atributos

Por domínio de um atributo, entendemos o conjunto de valores que o atributo pode assumir durante o seu ciclo de vida. Com base neste domínio, definiremos as características físicas de armazenamento do atributo dentro da relação. Os domínios representam apenas atributos atômicos. Quando o atributo é composto, há muita dificuldade em tornar claro o domínio de cada um deles.

Para definirmos o domínio de cada atributo precisamos conhecer o conjunto de valores que ele pode assumir ou receber. Vamos considerar alguns atributos básicos, para exemplificar o domínio de cada um deles:

- Idade: este atributo deverá possuir apenas valores inteiros, e sem sinal (positivos), maiores ou iguais a zero (este é o domínio da idade);
- Telefone: no Brasil, os números de telefones podem ser representados pelo número do DDD, seguido de 8 ou 9 dígitos. Assim, podemos afirmar que o domínio do telefone pode ser formado por 11 dígitos (de 0 até 9), para telefones no país.
- Salário: os salários dos funcionários devem ser armazenados com valores que possuam casas decimais, sendo duas delas suficientes. Podemos, então, afirmar que este atributo terá um domínio real com duas casas decimais.
- Nascimento: este atributo deverá conter uma data, sendo este o domínio do atributo e, por consequência, um controle que deveremos fazer, quando do armazenamento.
- Nome: o nome de uma pessoa será armazenado com caracteres, mas devemos determinar qual será o tamanho máximo de caracteres, que este atributo deverá conter.

Desta mesma forma, cada um dos nossos atributos prescinde de um estudo prévio sobre sua natureza, mas devemos também considerar o conjunto de valores que ele irá

assumir e em qual circunstância. Por exemplo, podemos citar o clássico "bug do milênio[8]". Nesta situação, o domínio para uma data havia sido estabelecido em circunstâncias limitadas (antes da virada do século) e, quando houve a necessidade de armazenar mais dois dígitos, ele não possuía espaço para isso.

f) Esquema do banco de dados

Por esquema do banco de dados relacional entendemos o conjunto de todas as relações que deverão fazer parte do banco, contendo cada um dos seus atributos. Não há a necessidade de definirmos o domínio de cada atributo no esquema, uma vez que já devemos conhecer cada domínio, individualmente. Por outro lado, precisamos indicar quais são, ou serão, as chaves primárias da relação.

A apresentação do esquema do banco acontece, normalmente, de duas formas diferentes. As duas são válidas, e você pode usar qualquer uma delas, sem problema ou prejuízo em função da outra. A primeira foi apresentada no final da Unidade 1 deste material. Nela, você descreve textualmente cada uma das relações, com os seus atributos nomeados, de forma resumida ou completa.

Nesta situação, devemos indicar o nome da relação e, dentro de parênteses, os seus atributos. Vejamos um exemplo de um esquema para uma pequena Clínica Médica:

- Médico (CRM, nome, telefone, (data-atendimento, hora-atendimento, código-paciente)).
- Paciente (código, nome, telefone, {convênio, descrição-convênio}).
- Consulta (CRM-Médico, Código-Paciente, Data, hora_consulta, valor_consulta, nome-convênio)

Observe que, neste exemplo, os atributos-chave (chaves primárias) estão sublinhados como forma de identificação. Existem, também, atributos compostos e multivalorados, os quais são apresentados dentro de chaves {}, indicando que podem ocorrer inúmeras vezes.

Uma segunda forma de apresentarmos o esquema do banco de dados é com o uso de tabelas. Neste caso, indicamos o nome da relação, fora da tabela, e seus atributos, internamente. Aqueles que são chave, devem ser sublinhados como antes. Veja um exemplo da mesma situação que a de cima:

8 Mais informações podem ser obtidas em: https://mundoeducacao.uol.com.br/informatica/bug-milenio.htm (visitado em junho/2020).

Médico

CRM	Nome	Telefone	Data-atendimento	Hora atendimento	Código paciente.

Paciente

Código	Nome	Telefone	Convênio	Descrição convênio

Consulta

CRM Médico	Código Paciente	Data	Hora consulta	Valor Consulta	Nome convênio

2.4 RESTRIÇÕES DE INTEGRIDADE

As restrições de integridade compõem um conjunto de regras e características que buscam manter o banco de dados íntegro, isto é, buscam prevenir danos acidentais ou propositais, os quais podem fazer com que a base de dados perca a sua finalidade de uso.

A integridade de dados é uma das características essenciais da segurança da informação, ao garantir que as informações não sofrerão alterações, as quais não foram autorizadas, ou que são impróprias.

2.4.1 RESTRIÇÕES DE INTEGRIDADE BÁSICA

Estas são restrições de integridade, fornecidas pelo próprio SGBD. Elas pertencem a dois grupos distintos: restrições de nulo e restrições de domínio.

a. Restrição de Nulo

Quando da criação da relação, no banco de dados, é possível definir que certo atributo não pode ser nulo. A partir deste ponto, o próprio banco de dados acusará uma situação de erro, assim que o atributo for armazenado com o valor vazio, e a operação não será permitida, o que torna este atributo obrigatório.

Por outro lado, é importante salientar que o valor nulo e o valor vazio podem ser

representados de forma distinta, em alguns SGBD's, e, por isso, os bancos podem não aceitar o valor explícito nulo (*null*), mas aceitarão um valor vazio para o campo.

b. Restrições de domínio

Este tipo de restrição é definido, quando da criação do atributo, no momento da implementação física do banco de dados. Ao definirmos o domínio, dentro das regras do banco que o atributo pode conter, automaticamente, estaremos eliminando elementos diferentes da representação.

Por exemplo, ao definirmos um campo como inteiro, estaremos eliminando armazenamento de caracteres nele, sendo também um elemento controlado pelo banco de dados, embora, normalmente, o banco não acuse um erro nestas situações. Ele dará uma advertência e buscará ajustar o valor inserido para algum contexto aceito pelo domínio. Por exemplo, ao armazenarmos um caractere no campo inteiro, a maioria dos bancos armazenará o valor zero naquele atributo.

Outro ponto relevante diz respeito aos elementos permitidos dentro dos diferentes domínios. Cabe ao programador verificar quais são os elementos disponíveis dentro do SGBD que foi escolhido para aquela implementação.

2.4.2 RESTRIÇÕES DE INTEGRIDADE DE ENTIDADE

Por definição, todas as tuplas de uma relação são distintas entre si. Para que isso aconteça, é necessário definirmos quais são as superchaves da relação. Podemos tratar como superchave o conjunto completo de atributos da relação e irmos refinando esta situação, até que tenhamos as chaves candidatas.

Do conjunto de chaves candidatas, acabamos escolhendo a chave primária da nossa relação. Ao indicarmos que um atributo, ou um conjunto deles, compõe a chave primária daquela relação, estamos impondo uma restrição de unicidade. Ou seja, a partir da indicação, aquele atributo deve ser único em todas as tuplas da relação, a chamada Restrição de Chave Primária, que garante a integridade da entidade como um todo.

Considere duas tuplas $\mathbf{t}_1$ e $\mathbf{t}_2$ de uma mesma relação e suponha que cada uma das tuplas tenha a sua PK (*primary key*), então uma restrição de integridade de chave impõe que $t_1[PK] \neq t_2[PK]$. Com isso, podemos usar o atributo para identificar a tupla que desejamos usar em certa operação e, ao banco de dados, não restará nenhuma dúvida à qual tupla estamos nos referenciando.

Além da questão de diferenciação, a restrição de integridade da entidade exige que nenhuma chave primária possa ter valor nulo.

2.4.3 RESTRIÇÕES DE INTEGRIDADE REFERENCIAL

Esta é, possivelmente, uma das restrições de integridade mais importantes no projeto de uma base de dados. Para definirmos tal tipo de restrição, precisamos retomar o conceito de Chave Estrangeira (FK). Uma chave estrangeira é um atributo da relação que indica uma chave primária de outra relação. Vamos a um exemplo prático. Suponha a relação Pedido, descrita a seguir:

Pedido (<u>número</u>, código cliente, código vendedor, data, valor total, forma pgto).

Na relação, o atributo 'número' é a sua chave primária, mas os dois atributos seguintes, o código do cliente que fez o pedido e o código do vendedor que o atendeu, são valores, os quais indicam tuplas de outras tabelas. Respectivamente, eles indicam tuplas das tabelas Cliente e Vendedor. Por conta disso, são chamadas de chaves estrangeiras, dentro da tabela Pedido.

Desse modo, considere um pedido qualquer, que tenha como código do vendedor, o número 15, e código do cliente, o número 23. Os valores devem, obrigatoriamente, indicar tuplas válidas e corretas de cada uma das suas tabelas.

As restrições de integridade referencial são usadas para manter a consistência entre duas tuplas de relações diferentes. Neste caso, uma tupla de uma relação que se refere a outra relação deve sempre referenciar uma tupla existente e válida nesta outra relação.

Voltando ao nosso exemplo, a integridade referencial quer dizer que as tuplas identificadas pela chave 15, na relação Vendedor, e 23, na relação Cliente, devem ser válidas e existentes.

Formalmente, dizemos que seja FK um conjunto de atributos de um esquema de uma relação R_1, definidos sobre o mesmo domínio dos atributos de chave primária PK de outro esquema R_2. Assim, para qualquer tupla t_1 de R_1 devemos ter:

a. $t_1[FK] = t_2[PK]$, onde t_2 é uma tupla de R_2; ou,
b. $t_1[FK]$ é nulo.

As restrições de integridade podem ser indicadas pela notação:

$$R_1[FK] \rightarrow R_2[PK]$$

Onde PK é uma chave primária da relação R_2, e FK é uma chave estrangeira da relação R_2, ambas com um mesmo domínio. Assim, em um esquema válido de banco de dados, podemos ter a seguinte representação:

Cliente (<u>código</u>, nome, endereço, bairro, CEP, telefone, e-mail)
Vendedor (<u>código</u>, nome, departamento)
Pedido (<u>número</u>, código cliente, código vendedor, data, valor total, forma pgto)
Pedido[código cliente] → Cliente[código]
Pedido[código vendedor] → Vendedor[código]

Partindo-se do princípio de que a chave primária de uma relação é única, podemos ter um grande problema, quando houver remoção das tuplas de uma relação. Por exemplo, o que irá acontecer, caso deletemos o cliente 23 da nossa tabela? Como ficam os seus pedidos?

É necessário tomar muito cuidado com operações do tipo: exclusão de tuplas e alteração na chave primária (se bem que isso não é comum). Dentro deste contexto, costuma-se indicar, junto com a declaração da chave estrangeira e da sua restrição de integridade, no esquema do banco de dados, incluir-se uma operação que deverá ser, obrigatoriamente, realizada em caso de alteração na relação, na qual está a chave primária.

As operações possíveis são o bloqueio, a propagação e a substituição por nulos. A opção de bloqueio faz com que a operação a ser realizada, na segunda relação onde está a chave primária, seja bloqueada, caso infrinja a restrição de integridade. Ou seja, caso a aplicação solicite a remoção do cliente, por exemplo, e este tenha pedidos no seu código, a remoção será negada pelo banco de dados.

A segunda opção de operação é a propagação. Neste caso, o banco de dados deverá propagar a operação para as tabelas em que estão as chaves estrangeiras. Por exemplo, na situação acima, na exclusão do cliente, os seus pedidos seriam excluídos na mesma operação.

Finalmente, a opção pela substituição por nulos, atribuirá o valor nulo para todas as chaves estrangeiras que deixarão de ser válidas, havendo a operação que causou o problema.

2.4.4 OPERAÇÕES NA BASE DE DADOS

Sobre este ponto, é importante mencionar que os problemas somente podem ser causados por operações de inserção, remoção (retirada ou exclusão), ou de modificação (alteração ou atualização) nos dados das tabelas. Podemos, então, indicar quais são as restrições que não podem ser violadas por cada uma delas. Vejamos:

= > Inserção

- Restrição de Domínio: valor fora do domínio.
- Restrição de Chave: valor já existe.
- Restrição de integridade de entidade: se chave for *null.*
- Restrição de integridade referencial: se chave estrangeira referencia uma tupla inexistente.
- Ação padrão: rejeitar inserção.

= > Remoção

- Restrição de integridade referencial: tupla deletada é referenciada por chaves estrangeiras
- Ação padrão: rejeitar remoção.
- Segunda opção: propagar remoção de tuplas que violem uma restrição de integridade referencial.
- Terceira opção: Modificar o valor da chave estrangeira para nulo.

= > Modificação

- Modificar o valor de um atributo que não é chave primária ou estrangeira não causa problemas (se o valor for do domínio e, se for null, que este valor seja permitido).
- Modificar a chave primária é igual a remover uma tupla e inserir outra.
- Modificar chave estrangeira: SGBD deve verificar se novo valor do atributo referência tupla é existente.

2.5 DEPENDÊNCIA FUNCIONAL

Segundo Elmasri e Navathe(2019), uma dependência funcional é definida como:

> "Uma dependência funcional, indicada por X → Y, entre dois conjuntos de atributos X e Y que são subconjuntos de R, especifica uma restrição sobre possíveis tuplas que podem formar um estado de relação r em R. A restrição é que, para quaisquer tuplas t_1 e t_2 em r, que tenham $t_1[X] = t_2[X]$, elas também devem ter $t_1[y] = t_2[Y]$." (ELMASRI, et al, 2019).

Isto quer dizer que os valores de Y de uma tupla em r dependem dos valores do componente X ou são determinados por eles. Estas dependências funcionais são restrições de integridade mais gerais que as restrições de chave, apresentadas acima. Por exemplo, podemos descrever as dependências:

CPF → Nome do cliente
Matrícula → {nome do funcionário, departamento}

Nos exemplos, lemos que o CPF determina o nome do cliente, dentro da nossa relação, assim como a matrícula determina, no nome do funcionário, o departamento ao qual ele está alocado. Na prática, quer dizer que, se duas tuplas tiverem o mesmo número de CPF, elas devem ter, obrigatoriamente, o mesmo nome de cliente. Assim, não serão válidas duas tuplas que não respeitem isto.

Obviamente, as dependências funcionais, como todas as restrições de integridade, são baseadas na semântica da aplicação. Desta forma, as dependências são consequência da análise de requisitos e do projeto lógico, realizados para a aplicação.

Estendendo o nosso projeto exemplo, podemos incluir as dependências:

Cliente (<u>código</u>, nome, endereço, bairro, CEP, telefone, e-mail)
Vendedor (<u>código</u>, nome, departamento)
Pedido (<u>número</u>, código cliente, código vendedor, data, valor total, forma pgto)
Pedido[código cliente] → Cliente[código]
Pedido[código vendedor] → Vendedor[código]
Código Cliente → {nome, endereço, bairro, CEP, telefone, e-mail}
Código Vendedor → {nome, departamento}
Número pedido → {código cliente, código vendedor, data, valor total, forma pgto}

Podemos, ainda, a partir de algumas dependências funcionais, inferir a existência de outras, como consequência destas. Por exemplo, suponha as dependências:

código produto → nome do produto

nome do produto → preço unitário

Se o código do produto determina o seu nome, e este o seu preço, podemos inferir que:

código produto → preço unitário

2.6 FORMAS NORMAIS

Durante o projeto do banco de dados, buscamos projetar as relações, de modo a obter o máximo de independência de dados, eliminando todas as redundâncias desnecessárias no conjunto destes dados.

Entendemos Formas Normais como um conjunto de regras, às quais uma tabela deve obedecer, para que as redundâncias sejam eliminadas. A passagem de uma tabela em cada uma das fases normais é o que define o processo chamado de Normalização, próximo tópico desta unidade.

Na literatura da área de banco de dados, existem cinco formas normais a serem aplicadas em uma base de dados. Na prática, exige-se que as relações de qualquer base estejam, no mínimo, na terceira forma normal.

Tipicamente, quando montamos o primeiro esquema de uma base de dados, as nossas relações são ditas como "Não Normalizadas" ou sem forma normal. Vamos considerar uma relação que se enquadra na situação: neste exemplo, vamos considerar uma relação que armazena dados de um Projeto e dos empregados que participam dele:

Projeto (Código_projeto, tipo_projeto, descrição, (Número_empregado, nome_empregado, categoria, salário, data_início, tempo_alocado))

Vamos observar uma possível instância desta relação, com informações que exemplificam os seus dados:

Código Projeto	Tipo Projeto	Descrição	Empregado					
			Número	Nome	Categoria	Salário	Data Início	Tempo
PR01	Desenvolvimento	Sistema de Controle de Vendas	10	Paula	A5	1200	10/02/18	6 h
			25	João	A1	1500	09/03/15	4 h
			32	José	B2	3200	20/12/17	10 h
			37	Mario	B3	2800	18/10/19	6 h
			47	Ana	A2	1400	12/06/18	8 h
			58	Maria	A3	2500	03/05/17	6 h
PR02	Manutenção	Sistema de RH	25	João	A1	1500	25/10/18	8 h
			40	Paulo	B2	1800	01/01/19	4 h
			58	Maria	A3	2500	15/07/18	12 h

É possível observarmos, na tabela, que há redundância dos dados. Observe, por exemplo, os empregados de número 25 e 58, os quais aparecem duas vezes na mesma relação. Portanto, há redundância nos dados.

Para caracterizar o motivo, pelo qual essa relação não é normalizada, precisamos primeiramente identificar cada uma das formas normais. Vejamos as características de cada uma delas:

- **Primeira Forma Normal (1FN)**

Uma relação está na Primeira Forma Normal, quando todos os atributos que a compõem são atômicos, ou seja, se todas as colunas que a compõem são atômicas e indivisíveis. Desta forma, todos os atributos devem conter apenas um valor correspondente, singular e não existem grupos de atributos repetidos.

- **Segunda Forma Normal (2FN)**

Uma relação está na Segunda Forma Normal se ela estiver na 1NF, e todo atributo não chave primária é plenamente dependente de toda a chave primária, não de apenas parte dela. Para que isso aconteça, devemos ter chaves primárias compostas. Caso a chave primária seja atômica, a relação já está na segunda forma normal.

- **Terceira Forma Normal (3FN)**

Uma relação está na Terceira Forma Normal, se ela estiver na 2FN, e nenhum dos atributos não chave é transitivamente dependente da chave primária. Na 3FN todos os atributos não chave devem depender diretamente da chave primária, sem que outro atributo qualquer se encontre no meio desta dependência.

Existe uma variação da 3FN, que é conhecida como BCNF ou Boyce/Codd Normal Form (Forma Normal de Boyce e Codd). Esta forma normal é um aperfeiçoamento da 3FN, destinada a situações em que se verifique a existência de mais de uma chave candidata, e que duas delas possuam elementos em comum. A BCNF é definida como: "Uma relação está na BCNF, quando todos os atributos são dependentes da chave, de toda a chave e de nada a mais do que a chave".

- **Quarta Forma Normal (4FN)**

Uma tabela está na Quarta Forma Normal, quando ela estiver na 3FN e não possuir dependências funcionais multivaloradas, ou seja, campos que se repetem em relação à chave primária, gerando redundância nas tuplas da entidade. Devemos fragmentar esta relação, com o objetivo de não termos mais as dependências funcionais do gênero.

- **Quinta Forma Normal (5FN)**

Uma tabela está na 5FN se, e somente se, estiver na 4FN, e um relacionamento triplo não puder ser decomposto em relacionamentos binários sem geração de informação incorreta.

As formas normais comportam-se de forma inclusiva, como demonstrado na Figura 11. Assim, estando uma relação em uma forma normal de grau mais alto, ela também estará, de modo obrigatório, nas formas normais inferiores a ela.

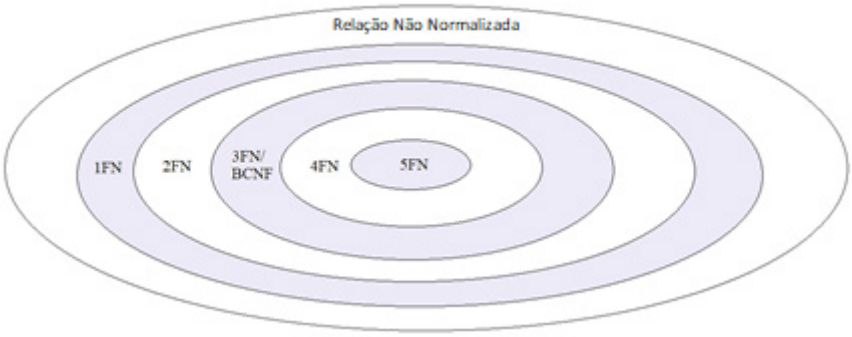


Figura 11 - Formas Normais.
Fonte: Elaborado pelo autor (Junho de 2020).

2.7 NORMALIZAÇÃO

Normalização é o processo de transformar relações não normalizadas em relações que se enquadrem em formas normais mais altas. É o processo de modelar o banco de dados, projetando a forma como as informações serão armazenadas, e eliminando, ou buscando eliminar, a redundância no banco. Este processo é feito, a partir da identificação de anomalia em uma relação, decompondo-a em relações melhor estruturadas.

Normalmente, durante o processo de análise, precisamos remover ou relocar uma ou mais colunas ou atributos de uma relação, dependendo da anomalia que identificamos. Estas anomalias estão ligadas diretamente às Formas Normais descritas na seção anterior. Assim sendo, o processo de normalização resume-se em passarmos as relações por cada uma das formas normais, até que se alcance a FN desejada. Tipicamente, exige-se, ao menos, que as relações estejam na terceira forma normal, mas graus maiores são desejáveis, a não ser que o ganho obtido com elas seja mínimo, perto do trabalho necessário para alcançá-las.

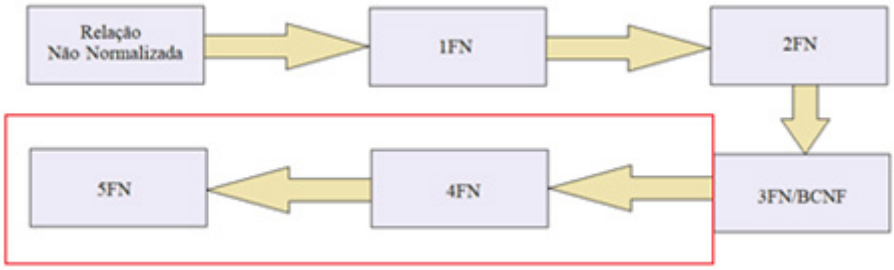


Figura 12 - Normalização.
Fonte: Elaborado pelo autor (Junho de 2020).

Para realizarmos a normalização, devemos seguir passos de análise nas relações, e ir garantindo, de forma gradativa, que uma relação passou por cada uma das fases, ou seja, primeiro garantimos que a relação está na 1FN. Feito isso, passamos para a 2FN, depois para a 3FN, e assim por diante.

Analisemos o processo de normalização com um exemplo prático. Suponha que temos uma tabela, dentro de uma planilha no computador, que contenha dados de projetos que sua empresa realiza, e dos empregados envolvidos em cada projeto. Por exemplo, veja uma instância desta planilha, na Figura 13.

CodProj	TipoProj	Descr	Emp					
			NoEmp	Nome	Cat	Sal	DataInicio	TempoAloc
LSC001	Novo Desenv	Sistema Estoque	2146	João	A1	400	01/11/91	24
			3145	Silvia	A2	400	02/10/91	24
			6126	José	B1	900	03/10/92	18
			8191	Mário	A1	400	01/11/92	12
			1214	Carlos	A2	400	04/10/92	18
PAG02	Manut.	Sistema RH	8191	Mário	A1	400	01/15/93	12
			4112	João	A2	400	04/01/91	24
			6126	José	B1	900	01/11/92	18

Figura 13 - Planilha exemplo.
Fonte: Elaborado pelo autor (Junho de 2020).

Com a planilha da Figura 13, podemos montar um esquema básico de relação, que é:

Projeto (Cod_Projeto, Tipo_Projeto, Descrição, (Num_empregado, Nome, Categoria, Salário, Data_Início, Tempo_Alocado))

Esta é uma relação que não está normalizada, podendo ser observado, na Figura 12 acima, quando identificamos várias linhas com a mesma informação. Isso também está no esquema acima, onde há um atributo multivalorado, representando o que foi vendido ao cliente.

Para fazermos a normalização desta relação, vamos verificar cada uma das formas normais, iniciando pela primeira delas.

- **Primeira Forma Normal (1FN)**

Uma tabela, pela definição, está na primeira forma normal, quando não houver nela nenhum atributo que não seja atômico. Em nossa relação exemplo, isso acontece com os empregados envolvidos nos projetos.

Para passar uma relação à primeira forma normal, pegamos cada uma das subtabelas ou tabelas embutidas e criamos com ela uma nova relação. Esta nova relação deve conter as chaves primárias da relação externa e as chaves da relação embutida. Ao final, identificamos quais serão as novas chaves primárias das relações formadas.

Assim, vamos separar os projetos dos dados dos empregados envolvidos neles.

Projeto (Cod_Projeto, Tipo_Projeto, Descrição, (Num_empregado, Nome, Categoria, Salário, Data_Início, Tempo_Alocado))

Projeto (Cod_Projeto, Tipo_Projeto, Descrição)
ProjetoEmpregado (Cod_Projeto, Num_empregado, Nome, Categoria, Salário, Data_Início, Tempo_Alocado)

Observe que esta simples separação não permite determinar qual foi o empregado que está atuando no projeto. Para isso, devemos incluir a chave primária da relação externa (primeira) nesta nova relação.

Como agora as duas relações não possuem subtabelas ou sub-relações dentro delas, dizemos que ela está na 1FN.

O resultado é:

Projetos

CodProj	Tipo	Descr
LSC001	Novo Desenv.	Sistema de Estoque
PAG02	Manutenção	Sistema de RH

Projeto Empregado

CodProj	CodEmp	Nome	Cat	Sal	DataIni	TempAl
LSC001	2146	Joao	A1	400	1/11/91	24
LSC001	3145	Silvio	A2	400	2/10/91	24
LSC001	6126	Jose	B1	900	3/10/92	18
LSC001	1214	Carlos	A2	400	4/10/92	18
LSC001	8191	Mario	A1	400	1/11/92	12
PAG02	8191	Mario	A1	400	1/05/93	12
PAG02	4112	Joao	A2	400	4/01/91	24
PAG02	6126	Jose	B1	900	1/11/92	12

- **Segunda Forma Normal (2FN)**

Na segunda forma normal, todas as relações devem ter apenas atributos, os quais sejam dependentes de toda a chave primária escolhida, e não de apenas parte dela. Com

isso, todas as relações que tenham uma chave primária, composta de um **único** atributo, já se encontram na 2FN. No esquema acima, a relação Projeto já se encontra neste caso. Devemos, então, analisar apenas as relações, contendo chaves primárias compostas de mais de um atributo e, além disso, pelo menos um atributo adicional que não seja chave, o que acontece na relação ProjetoEmpregado. Vamos a ela:

ProjetoEmpregado (Cod_Projeto, Num_empregado, Nome, Categoria, Salário, Data_Início, Tempo_Alocado)

Observe que os atributos que não são chave: nome, categoria, salário, data_início e tempo_alocado dependem da chave. A questão é se eles dependem de toda a chave ou apenas de parte dela. Vejamos:

- Nome: depende **apenas de parte** da chave (Num_Empregado)
- Categoria: depende **apenas de parte** da chave (Num_Empregado)
- Salário: depende **apenas de parte** da chave (Num_Empregado)
- Data_Início depende de **toda** a chave (início do empregado no projeto)
- Tempo_Alocado depende de **toda** a chave (tempo do empregado no projeto)

Em função da análise quanto à dependência funcional, precisamos separar esta relação em duas outras. Uma delas conterá os atributos que relacionam o empregado ao projeto, e a outra, os dados do empregado.

ProjetoEmpregado (Cod_Projeto, Num_empregado, Nome, Categoria, Salário, Data_Início, Tempo_Alocado)

Projeto (Cod_Projeto, Tipo_Projeto, Descrição)
ProjetoEmpregado (Cod_Projeto, Num_empregado, Data_Início, Tempo_Alocado)
Empregado (Num_empregado, Nome, Categoria, Salário)

É possível perceber, olhando para a instância que foi usada como exemplo, que os dados ficaram mais organizados agora. Vejamos:

Projetos

CodProj	Tipo	Descr
LSC001	Novo Desenv.	Sistema de Estoque
PAG02	Manutenção	Sistema de RH

ProjetoEmpregado

CodProj	CodEmp	DataIni	TempAl
LSC001	2146	1/11/91	24
LSC001	3145	2/10/91	24
LSC001	6126	3/10/92	18
LSC001	1214	4/10/92	18
LSC001	8191	1/11/92	12
PAG02	8191	1/05/93	12
PAG02	4112	4/01/91	24
PAG02	6126	1/11/92	12

Empregado

CodEmp	Nome	Cat	Sal
2146	Joao	A1	400
3145	Silvio	A2	400
6126	Jose	B1	900
1214	Carlos	A2	400
8191	Mario	A1	400
4112	Joao	A2	400

- **Terceira Forma Normal (3FN)**

Chegamos à forma normal mais usual nos nossos projetos. Na 3FN, pede-se que cada atributo não chave dependa diretamente da chave, e não de forma transitiva, ou seja, passando por outro atributo antes. Assim sendo, toda relação que contenha apenas um atributo não chave já estará na 3FN. Precisamos analisar apenas as relações com dois ou mais atributos nesta situação.

A princípio, precisamos determinar se há, na relação, dependências transitivas ou indiretas da chave principal. Vamos analisar cada uma das relações que estão na 2FN. Em primeiro lugar, a relação Projeto:

Projeto (Cod_Projeto, Tipo_Projeto, Descrição)

Observe que os campos não chave são o Tipo_Projeto e a Descrição. Podemos afirmar que ambos dependem diretamente do Cod_Projeto:

Cod_Projeto → Tipo_Projeto
Cod_Projeto → Descrição

Desta forma, a relação está na 3FN. Tomemos, agora, a relação ProjetoEmpregado:

ProjetoEmpregado (Cod_Projeto, Num_empregado, Data_Início, Tempo_Alocado)

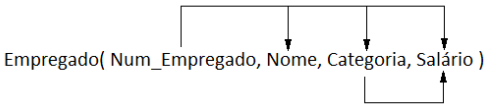
Nesta relação, os atributos a serem analisados são: Data_Início e Tempo_Alocado. Perceba que ambos estão relacionados ao trabalho do empregado naquele projeto. Portanto, ambos dependem da chave diretamente, e a relação está na 3FN:

{Cod_Projeto, Num_Empregado) → Data_Início
{Cod_Projeto, Num_Empregado) → Tempo_Alocado

Finalmente, vamos tomar a relação de Empregados:

Empregado (Num_empregado, Nome, Categoria, Salário)

Nela, facilmente percebemos que o Nome é dependente do Num_Empregado, assim como a categoria, mas vamos considerar que o salário está relacionado com a categoria, antes de chegar ao empregado. Veja:



Desta forma, a situação é a seguinte:

Num_Empregado → Nome
Num_Empregado → Categoria → Salário

Em função desta dependência indireta, precisamos separar a relação Empregado em

duas outras. Uma delas ficará com os dados do empregado, e a outra com os dados da categoria. Para que isso funcione, precisaremos incluir na relação Empregado uma chave estrangeira que indica a sua categoria. Ao final, o resultado é:

Projeto (Cod_Projeto, Tipo_Projeto, Descrição)
ProjetoEmpregado (Cod_Projeto, Num_Empregado, Data_Início, Tempo_Alocado)
Empregado (Num_Empregado, Nome, Cod_Categoria)
Categoria (Cod_Categoria, Categoria, Salário)

- **Forma Normal Boyce-Codd (BCNF)**

Esta forma normal foi proposta como uma simplificação da 3FN, porém se mostrou mais rígida que a primeira. Portanto, uma relação que estiver na BCNF estará na 3FN, mas o contrário não é sempre verdade. Ela diz que "Uma relação está na BCNF, quando todos os atributos estão dependentes da chave, de toda a chave e de nada mais do que a chave". Veja que somente a parte final da última definição da 3FN não está presente: "A é um membro de alguma chave".

A definição original da 3FN não lida adequadamente com uma relação que:

- Tenha duas ou mais chaves candidatas;
- Estas chaves candidatas sejam compostas;
- Estas chaves sejam sobrepostas, ou seja, tenham atributos em comum.

Caso isso não aconteça nas suas relações, basta aplicar a 3FN nelas. Por outro lado, se houver mais chaves candidatas, devemos analisar a relação. A definição da BCNF diz que uma relação está nela, sempre que uma dependência funcional não-trivial X A se mantiver na relação.

Dependências funcionais triviais (aquelas que são satisfeitas por todas as instâncias de tabela possíveis). Desta forma, as dependências triviais têm a forma X -> Y, onde Y está contido em X. Por exemplo:

{Nome, Endereço} → {Nome}
{Nome} → {Nome}

Logo, uma relação está na BCNF, se todas as suas dependências funcionais não triviais forem da forma **superchave -> conjunto-de-atributos**. Em outras palavras, além das dependências funcionais triviais, só podemos ter restrições de chave. Uma relação na BCNF está livre de redundâncias, que podem ser detectadas usando dependências funcionais.

Para entendermos isso, considerar-se-á uma pequena modificação em nosso esquema exemplo. Vamos supor que um funcionário pode estar enquadrado em mais de uma categoria e receba o valor de todas elas como salário. Assim, teríamos uma situação, antes da 3FN, como:

Num_Empregado	Nome	Categoria	Salário
2	João	A1	300,00
2	João	A2	500,00
3	Ana	B1	400,00
.....	....	....	...

Observe o que poderíamos usar como chave na tabela. Lembre-se de que chave é algo que não pode ser repetido na relação. Podemos usar o {Num_Empregado,Categoria} ou {Nome, Categoria}. Temos chaves compostas e com atributos sobrepostos, neste caso, a Categoria.

Se aplicássemos a 3FN nesta relação, ela estaria normalizada com o mesmo processo do caso anterior, contudo não neste caso, pois haveria redundância. Para resolver, será preciso criar duas outras relações, uma para armazenarmos as Categorias e os salários, como no caso anterior, além de outra, para armazenar o relacionamento entre cada Empregado e as suas Categorias:

Empregado (Num_Empregado, Nome)
Categoria (Cod_Categoria, Categoria, Salário)
EmpregradoCategoria (Num_Empregado, Cod_Categoria)

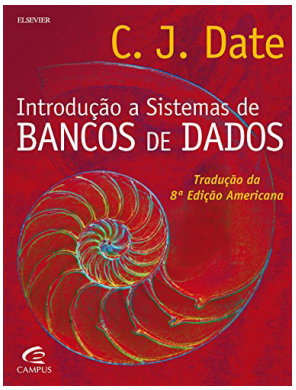
Com essa nova estrutura, o esquema estará na 3FN e, também, na BCNF.

Conforme mencionado anteriormente, trazer uma relação até a 3FN/BCNF é o mínimo necessário, para uma implementação com redundância mínima. Já que as formas normais 4FN e 5FN demandam serviço adicional, com um menor ganho na implementação, nosso processo de normalização estará finalizado no ponto atual.

SAIBA MAIS

Saiba mais sobre o processo de modelagem, nos vídeos do grupo: https://www.youtube.com/watch?v=Q_KTYFgvu1s&list=PLucm8g_ezqNoNHU8tjVeHmR-GBFnjDIlxD

SUGESTÃO DE LIVRO



O livro Introdução a Sistemas de Banco de Dados, de autoria de C. J. Date é uma das principais referências na área de banco de dados. Todos os conceitos apresentados nesta unidade são expostos em detalhes nesta obra.

CONSIDERAÇÕES FINAIS

Na segunda unidade, começamos a analisar o projeto lógico de um banco de dados e suas necessidades funcionais. Em primeiro lugar, definimos as características elementares do banco, como chaves e restrições.

Com base nas restrições e suas consequências, partimos de um modelo ou esquema do banco de dados, para apresentar os recursos necessários à qualificação do projeto, minimizando a redundância e otimizando o uso desta base nas aplicações. Isto é realizado, fundamentalmente, com a aplicação do conceito de dependência funcional, para avaliarmos as diferentes formas normais, a serem aplicadas nas relações do banco de dados. Por mais que esta unidade apresente cinco diferentes de formas, mais a variação BCNF, em uma aplicação prática, vamos até a 3FN/BCNF. As formas normais 4 e 5 são importantes em situações bem específicas, principalmente, quando o ganho na sua aplicação é inferior ao trabalho necessário para a aplicação/confirmação.

Partindo-se de uma base/esquema organizado e otimizado, podemos iniciar o processo de desenvolvimento. Na próxima unidade, vamos analisar os caminhos de projeto que nos levam até o esquema, o qual foi otimizado aqui.

EXERCÍCIO FINAL

1. No contexto de bancos de dados relacionais, particularmente em relação à normalização, considere as seguintes afirmações:

I- Uma relação está na Primeira Forma Normal (1FN), quando ela não contém tabelas aninhadas;

II- Uma relação está na Segunda Forma Normal (2FN), quando, além de estar na Primeira Forma Normal, não contém dependências parciais;

III- Uma relação está na Terceira Forma Normal (3FN) se, e somente se, além de estar na Segunda Forma Normal, não possuir dependências funcionais transitivas ou indiretas.

Levando-se em conta as três afirmações acima, identifique a única alternativa válida.

a) Apenas a I e a II estão corretas.

b) Apenas a II e a III estão corretas.

c) Apenas a I e a III estão corretas.

d) As afirmações I, II e III estão corretas.

e) As alternativas I, II e III estão incorretas.

2. Considere:

I. Conjunto de um ou mais atributos que, quando tomados coletivamente, permitem identificar exclusivamente uma entidade.

II. Restrição que especifica que o atributo é uma chave candidata (tem um valor exclusivo para cada linha de uma tabela).

Correspondem, respectivamente, a I e II:

a) chave única e chave estrangeira.

b) chave primária e chave estrangeira.

c) chave primária e chave única.

d) chave estrangeira e chave primária.

e) chave estrangeira e chave única.

3. Seja a relação EMP-PROJ(CPF, NumProj, Horas, NomeEmp, NomeProj, LocalProj) onde {CPF, NumProj} é a chave primária de EMP-PROJ e as seguintes dependências funcionais:

{CPF, NumProj} → Horas

{CPF} → NomeEmp

{NumProj} → {NomeProj, LocalProj}

A relação EMP-PROJ, com estas dependências funcionais, viola qual forma normal?

a) Primeira Forma Normal.

b) Segunda Forma Normal.

c) Terceira Forma Normal.

d) Parcialmente, entre a segunda e terceira forma normal.

e) Não normalizada.

4. Considere as tabelas a seguir, com seus respectivos campos (PK são as chaves primárias).

TabelaA = { codigo_projeto (PK),codigo_funcionario (PK), nome_funcionario, cargo_funcionario, data_inicio_projeto, duração_projeto}
TabelaB = {codigo_departamento (PK), nome_departamento, telefone_departamento, localizacao}
TabelaC = {cpf_cliente (PK), codigo_produto (PK), nome_cliente, valor_unitario_produto, qtdade_produto}

TabelaD = {numero_processo (PK), ano_processo, local_processo, vara }
TabelaE = {ra_aluno (PK), codigo_disciplica (PK), nome_aluno, carga_horaria_disciplina, nota}

Durante um processo de normalização, deveremos verificar se há violação da Segunda Forma Normal, apenas nas tabelas:

a) Tabela B e Tabela D.

b) Tabela A e Tabela E.

c) Tabela A, Tabela B e Tabela C.

d) Tabela C e Tabela E.

e) Tabela A, Tabela C e Tabela E.

5. Analise as sentenças abaixo, as quais discutem as principais restrições que podem ser expressas no modelo de dados relacional.

I. Restrições de domínio são impostas, para garantir que os valores nas colunas sejam atômicos e que respeitem os tipos de dados das colunas.

II. Restrição de integridade de entidade diz respeito à garantia de que toda linha em uma tabela deve ser única.

III. A restrição de integridade referencial garante que determinadas colunas em uma tabela sejam iguais às colunas que compõem a chave primária de outra tabela ou da própria tabela.

IV. A restrição de chave estabelece que nenhum valor de chave primária pode ser vazio.

Estão CORRETAS as afirmativas:

a) I e III.

b) II e III.

c) I e IV.

d) III e IV.

e) II e IV.

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados**. 1. ed. São Paulo: Érica, 2014.

AMADEU, Cláudia (org). **Banco de Dados**. São Paulo: Pearson, 2014.

BARBOZA, Fabrício. **Modelagem e Desenvolvimento de Banco de Dados**. Porto Alegre: SAGAH, 2018.

CARDOSO, Virginia. CARDOSO, Giselle. **Sistemas de Banco de Dados: uma abordagem introdutória e prática**. São Paulo: Saraiva, 2012.

CARDOSO, Virginia; CARDOSO, Giselle. **Linguagem SQL**: **fundamentos e práticas**. São Paulo: Saraiva, 2013.

DATE, Chris. Introdução a Sistemas de Banco de Dados. 8.ed. São Paulo, Campus: 2004.

ELSMARI, Ramez. NAVATHE, Shamkant**. Sistemas de Banco de Dados**. 7. ed. São Paulo: Pearson, 2019.

HEUSER, Carlos. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

MACHADO, Felipe. Banco de Dados: Projeto e Implementação. 4. ed. São Paulo: Érica, 2014.

MEDEIROS, Luciano. **Banco de dados: princípios e práticas**. Curitiba: Intersaberes, 2013.

MANNINO, Michael V. **Projeto, desenvolvimento de aplicações e administração de banco de dados**. 3. ed. Porto Alegre: AMGH, 2014.

RAMARKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados**. 3. ed. Porto Alegre: AMGH, 2011.

TEOREY, Toby. et al. **Projeto e Modelagem de Banco de Dados**. 2. ed. São Paulo: Elsevier, 2014.

unidade 3

MODELAGEM DE DADOS

3.1 INTRODUÇÃO À UNIDADE

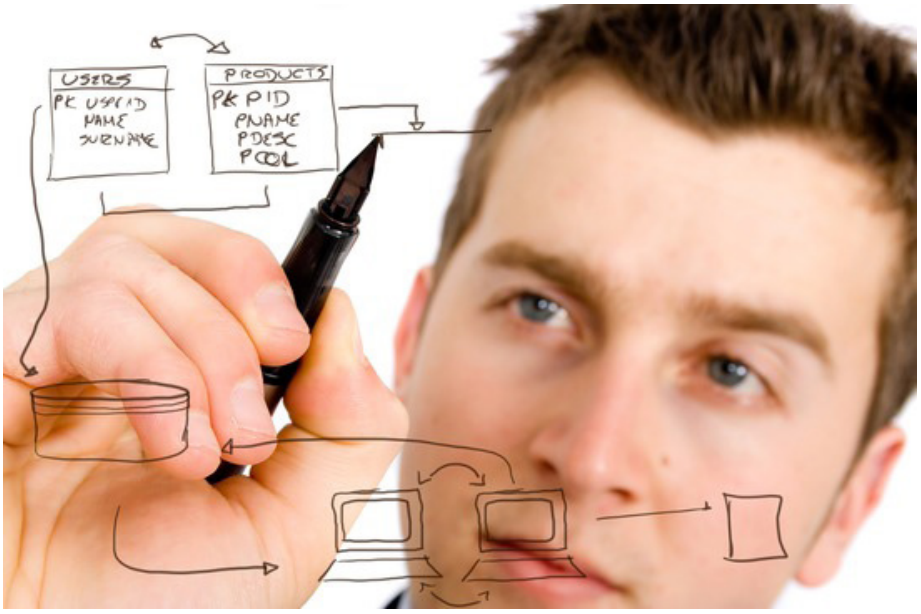


Figura 14 - Modelagem de Dados.
Fonte: <http://aprendaplsql.com/modelagem-de-dados/modelagem-de-dados-parte-01/> Acesso em: Junho de 2020.

Bem-vindos!

Em nossa terceira unidade, iniciaremos o processo de modelagem das bases de dados. Após a análise de requisitos, o próximo passo, inclusive feito antes da normalização em algumas situações, é fazermos o modelo lógico da nossa base. Para isso, existem ferramentas de modelagem que auxiliam a descrição do que deve ser implementado.

Primeiramente, analisaremos a modelagem feita com o Modelo ER, criado por Peter Chen, no início dos anos 1970, usado até hoje. Este é um modelo de alto nível que permite a compreensão do projeto pelos usuários finais e, por conta disso, é muito útil no envolvimento dos usuários com o projeto do banco.

Na segunda seção desta unidade, vamos estender o Modelo ER, para incluir nele elementos do paradigma da orientação a objetos, o que permite "atualizar" este processo de modelagem às dinâmicas mais atuais de projeto de software.

Finalmente, para encerramos a unidade, apresentaremos, detalhadamente, o diagrama de classes da Linguagem UML, mas com o foco em uma ferramenta de modelagem para o projeto de banco de dados.

Os principais objetivos de aprendizagem da Unidade 03 são: conhecer o processo de modelagem de banco de dados e sua transformação em um modelo físico; transformar elementos da análise de requisitos em entidades de um banco relacional; compreender

os diferentes relacionamentos entre as entidades, e como apresentá-los no processo de modelagem; definir um esquema de banco de dados, derivado do processo de modelagem.

3.2 CONCEITOS BÁSICOS

Em um processo de modelagem em banco de dados, existem vários níveis de abstração, como pode ser observado na Figura 15. Partindo-se da situação real (minimundo), é elaborado um projeto conceitual que descreve a estrutura do banco de dados, de forma independente do SGBD que foi escolhido. Este é o modelo, no qual os objetos e suas características e relacionamentos são representados de forma fiel àquilo que foi observado, podendo-se abstrair e compreender melhor o ambiente.

Logo após, partindo-se do modelo conceitual, é elaborado um modelo lógico, que representará a estrutura dos dados, no banco de dados, o qual está ligado ao modelo de banco escolhido, cujos objetos, características e relações são representados, de acordo com as técnicas que escolhermos. Ainda estamos em uma fase que é independente dos meios de armazenamento escolhidos, mas somos dependentes do modelo de banco que foi adotado.

Finalmente, é elaborado o modelo físico, no qual se determina como será a implementação física do modelo no SGBD a ser utilizado. Aqui, a representação dos objetos é implementada na estrutura final de representação.

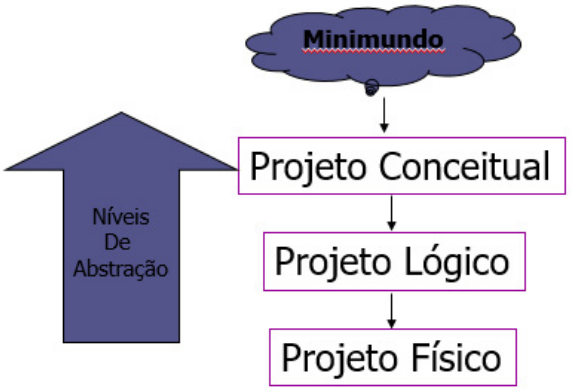


Figura 15 - Níveis de abstração nos projetos do banco de dados.
Fonte: Elaborado pelo autor (Junho de 2020).

Nesta unidade, vamos tratar dos modelos conceituais que podem ser usados na construção do nosso banco, Mas, o que é um modelo? Segundo DATE (2004), é uma representação abstrata e simplificada de um sistema real onde podemos explicar ou testar

o seu comportamento. Já por modelo de dados, o mesmo autor define a representação desta realidade, através de algumas regras e símbolos que fazem a abstração do mundo real, representando o modelo de negócio da empresa ou do sistema, através de uma técnica de modelagem.

Na construção deste tipo de modelo, precisamos definir, antecipadamente, alguns elementos importantes. Em primeiro lugar, decidimos qual a abrangência ou o escopo do modelo, isto é, o que ele irá conter e, inclusive, o que não conterá. Em seguida, definimos o seu nível de detalhamento. Finalizamos, definindo qual será o tempo e os recursos que podemos utilizar no processo.

Pelo exposto, podemos entender que a etapa de modelagem faz parte do planejamento, necessário à construção do banco de dados. Então, dizemos que a execução da modelagem de dados tem objetivos claros no processo de planejamento. São eles:

a. Representar um ambiente observado ou o minimundo.
b. Servir de instrumento para comunicação.
c. Favorecer o processo de verificação e validação.
d. Capturar aspectos de relacionamentos entre os objetos observados.
e. Servir como referencial para a geração de estruturas de dados.
f. Estabelecer conceitos únicos, a partir de várias visões.

Para executarmos esse processo, podemos definir alguns passos importantes. São eles:

a. Observação dos objetos, de acordo com o nível de abrangência e de detalhamento que escolhemos ao modelo.
b. Para representarmos corretamente cada um dos objetos, antes devemos identificá-lo e compreendê-lo, sendo este o segundo passo.
c. Após a identificação e compreensão dos objetos, devemos aplicar a técnica de modelagem escolhida para representar suas características, relacionamento e comportamentos.
d. Verificamos, então, a fidelidade e a coerência do modelo que montamos no passo anterior. Caso existam falhas, retomamos o processo nos passos anteriores, conforme a necessidade.
e. Finalmente, devemos validar o modelo criado, normalmente, envolvendo os usuários finais para definirmos que ele é adequado à total representação do domínio desejado.

3.3 MODELO ENTIDADE-RELACIONAMENTO

A técnica de modelagem conceitual de banco de dados foi proposta por Peter Chen (1976 apud ALVES, 2014), como um modelo de dados conceitual de alto nível, cujos conceitos foram projetados para estarem mais próximos da visão que o usuário final tem dos seus dados, não se preocupando em representar o seu modo de armazenamento no banco de dados.

A construção deste modelo se baseia na observação de que, no mundo real do problema, podemos ter vários elementos com características próprias, que o definem, e que se relacionam, de acordo com certas regras e informações. O MER, ou Modelo Entidade-Relacionamento, baseia-se em três elementos fundamentais: entidades, atributos e relacionamentos. Vamos a eles!

3.3.1 ENTIDADES

Entidades reproduzem o conjunto de objetos e conceitos do mundo real, sobre os quais desejamos manter informações no banco de dados. Estas entidades realizarão conexões com outras, dentro do modelo, de forma a dar representação às diferentes situações e demandas, as quais devem ser demonstradas.

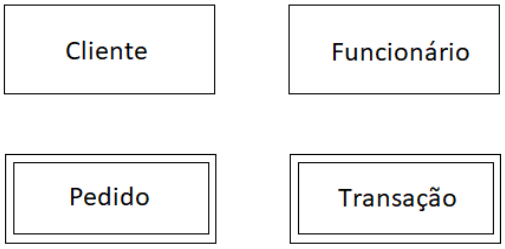
Existem dois tipos de entidades: as físicas e as conceituais. Entidades físicas são aqueles objetos ou elementos que podemos observar, fisicamente, no ambiente estudado. Por exemplo, podemos observar entidades físicas, como clientes, fornecedores, veículos, equipamentos e vários outros. As entidades conceituais não existem fisicamente, mas representam relações e elementos presentes nas regras de negócio que devemos modelar. Elas fazem sentido apenas dentro do domínio estudado, e não fora dele. Podemos citar, como exemplo, algum relatório, um pedido, tipos de clientes, usuários do sistema e outros. Fazendo uma analogia, podemos dizer que as entidades são equivalentes aos objetos da Linguagem UML e são (ou serão) as relações dentro do modelo relacional do banco de dados.

As entidades podem ser classificadas da seguinte forma:

a. **Entidades Fortes:** são entidades que se justificam apenas pela sua existência, ou seja, não dependem de nenhuma outra entidade para existirem. Por exemplo, podemos ter entidades como: Cliente, Fornecedor e outros.

b. **Entidades Fracas:** uma entidade fraca depende da existência de outra entidade, para que ela possa existir. Por exemplo, considere que definimos uma entidade em nosso modelo, para representar os dependentes de um funcionário. Neste caso, Funcionário é uma entidade forte, mas a entidade Dependente não é; ela é fraca, pois somente existirá, se houver a entidade funcionário para justificá-la.
c. **Entidades Associativas:** são criadas para representar os relacionamentos entre as entidades do modelo. Isso acontece, pois existem associações que não podem ser representadas apenas por uma chave estrangeira (atributo) em um modelo.

Cada entidade, no MER, é representada por um retângulo, contendo o nome dela dentro do modelo. As entidades fortes são representadas por um único retângulo, enquanto as fracas podem ser distinguidas por um retângulo de linhas duplas. Este nome pode (e deve) ser utilizado para indicar que tipo de elemento está sendo exposto ali. Podemos ter, então, entidades como as do nosso modelo:



3.3.2 ATRIBUTOS

Atributos são os tipos de dados, associados a uma entidade ou a um relacionamento dentro do modelo. Aqui, o conceito é o mesmo que dentro do modelo relacional, representando cada um dos tipos de dados que devemos manter para aquela entidade/relacionamento.

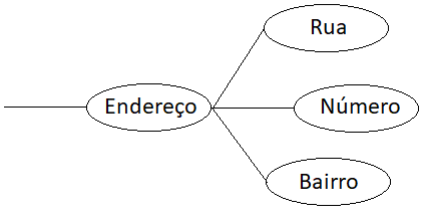
Assim como no modelo relacional, existem tipos diferentes de atributos dentro do modelo. Cada um dos tipos deve ser representado com um símbolo diferente. Os tipos possíveis são (observe os símbolos usados para cada um deles):

- Atributo normal: representa a maioria dos atributos da entidade.

- Atributo chave: designa a chave primária daquela entidade. Existem dois tipos, um para as chaves das entidades fortes (sublinhado com linha cheia) e outro para as chaves das entidades fracas (sublinhado com linha pontilhada).

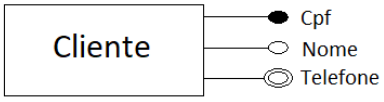
- Atributo multivalorado: são os atributos formados por mais de um valor. Por exemplo, o telefone, em que uma mesma entidade Cliente pode ter vários telefones.

- Atributo composto: são os atributos que podem ser divididos em partes menores (que serão atributos também).



- Atributo derivado: é um atributo, cujo valor pode ser obtido, a partir de outros atributos ou dos relacionamentos. Por exemplo, podemos ter um atributo idade (que poderia ser obtido, a partir da data de nascimento), ou número de funcionários (que pode ser obtido contando o número de cadastros).

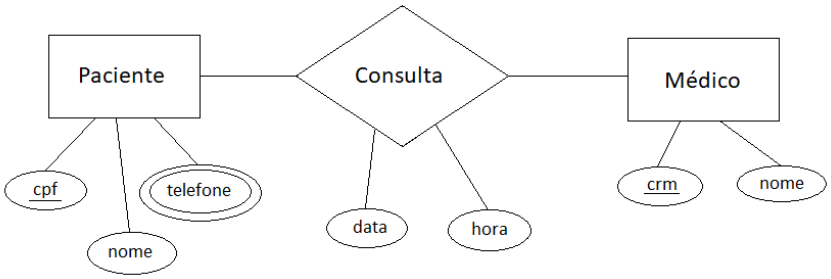
- Para simplificar, podemos representar os atributos com a descrição dos seus nomes fora da região fechada. A única mudança, neste caso, é indicarmos as chaves com uma região com o fundo escuro. Veja um exemplo:



3.3.3 RELACIONAMENTOS

O relacionamento é uma estrutura do Modelo ER que representa a relação entre duas ou mais entidades. Normalmente, ele indica uma ação que associa os elementos destas entidades.

O relacionamento é representado no modelo, através de um losango, contendo o nome da relação existente entre os elementos ligados por ele. Por exemplo, podemos ter uma relação chamada consulta, a qual liga as entidades Paciente e Médico. Veja:



É importante observar que um relacionamento, consulta neste caso, pode conter atributos, os quais são usados para indicar as informações que regulam cada instância desta relação. Quando um relacionamento possui atributos, ele deverá, necessariamente, ser representado com uma relação ou tabela no banco de dados.

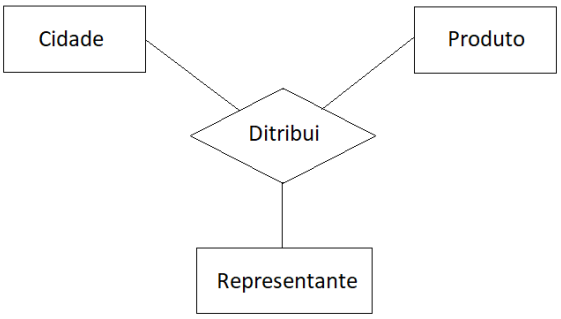
Existem dois símbolos, para representar um relacionamento, embora usemos o primeiro deles, apresentado acima, por simplicidade. Um losango simples indica um relacionamento sem identificação, enquanto o losango com linhas duplas indica um relacionamento com identificação. O relacionamento com identificação é representado por uma chave estrangeira, a qual é parte da composição da chave primária da entidade de referência.

Considere uma situação de exemplo, com as entidades Livro, Pessoa e Autor. Haverá um relacionamento simples, sem identificação, entre uma Pessoa e o Livro, pois ele pode existir sem a pessoa e pode, inclusive, mudar de proprietário. Já o relacionamento entre o livro e o autor (ou autores) é identificado, porque o livro precisa de um autor e não pode existir sem ele.

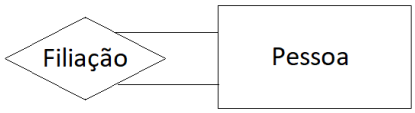
Existem três tipos de relacionamentos:

a) Binário: é o relacionamento entre duas entidades do modelo, apenas. O relacionamento acima, entre Paciente e Médico, é do tipo binário, pois liga apenas duas relações.

b) Ternário: é o relacionamento que liga três entidades diferentes. Veja uma situação exemplo, em que existe um representante para determinados produtos, e em cidades específicas. Observe:



c) Autorrelacionamento: é o relacionamento de uma entidade com ela mesma. No seguinte exemplo, temos uma pessoa, indicando quem são os seus pais (que também são pessoas).



Além de representarmos os relacionamentos entre as entidades, precisamos representar um conceito fundamental do MER, que é a indicação da cardinalidade das relações. **Cardinalidade** especifica quantas ocorrências de uma entidade podem estar associadas a uma determinada ocorrência da outra entidade.

Existem dois tipos de cardinalidade: **mínima** e **máxima**. A cardinalidade máxima expressa o número máximo de ocorrências de determinada entidade, associada a uma ocorrência da entidade em questão, através do relacionamento. A cardinalidade mínima expressa o número mínimo de ocorrências de determinada entidade, associada a uma ocorrência da entidade em questão, através do relacionamento. Para indicar estas cardinalidades, usamos a seguinte notação:

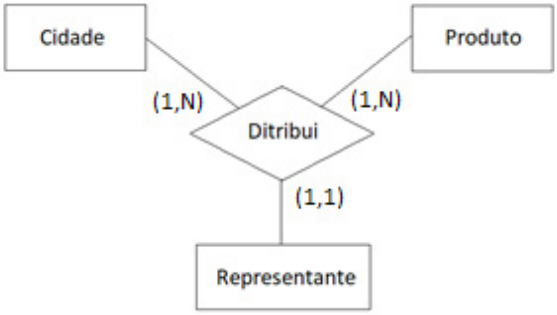
(número mínimo, número máximo)

Para indicar a cardinalidade, em todos os casos, utilizam-se os valores 0, 1 e N. Zero (0) indica a inexistência, ou não obrigatoriedade, da relação; um (1) indica que deve acontecer uma única ocorrência da relação para com aquela entidade; e N indica a multiplicidade do relacionamento. Vejamos um exemplo deste tipo de indicação:

Nesta situação, verificamos que existem duas indicações de cardinalidade, dos dois lados do relacionamento. Assim, para lermos, partimos de uma entidade, observando a sua cardinalidade no lado oposto. Por exemplo, a relação do Funcionário com o Dependente é (0,N). Já a relação entre o Dependente e o Funcionário é (1,1). Vamos interpretar isso:

- (0,N): indica que uma ocorrência de funcionário pode não estar associada a uma ocorrência de dependente, ou pode estar associada a várias ocorrências dele (determinado funcionário pode não possuir dependentes ou pode possuir vários);
- (1,1): indica que uma ocorrência de dependente está associada a apenas uma ocorrência de funcionário (determinado dependente possui apenas um empregado responsável).

Um problema desta situação de análise está presente nos relacionamentos ternários. Como devemos olhar para eles? Escolhemos uma dupla de entidades e as relacionamos com uma terceira. Veja, por exemplo:



Aqui, vamos tomar as duplas. Em primeiro lugar, pegamos o par Cidade-Representante. A cardinalidade deles em relação aos Produtos está indicada para o lado do Produto, ou seja, é (1,N). Isso quer dizer que, em uma cidade, o seu representante deve ter, pelo menos, um produto a oferecer, mas poderá ter vários. Já tomando o par Representante-Produto, sua relação com cidade está ao lado da Cidade. Ela indica que, em cada cidade, deve haver um produto a oferecer pelo representante, mas podemos ter vários. Finalmente, para Cidade-Produto, quando se relaciona com o Representante, temos a cardinalidade (1,1), indicando que deve haver, obrigatoriamente, só um representante para aquele produto, naquela localidade.

Desta forma, percebemos que a cardinalidade mínima é usada para indicar a obrigatoriedade da relação, isto é, se ela precisa existir ou não. Quando indicamos o valor 0 (zero), estamos dizendo que ela é opcional (o funcionário não precisa ter um dependente). Já a indicação de 1 (um), faz com que ela precise, necessariamente, ocorrer (um dependente precisa estar ligado a um funcionário, caso ele exista).

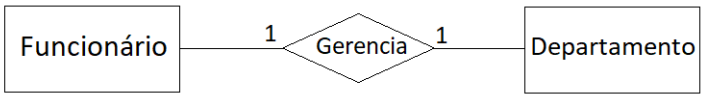
Podemos, caso seja conveniente, usar apenas a cardinalidade máxima em nossos modelos. Por outro lado, a indicação de ambas as situações permite uma melhor interpretação do modelo criado.

Quando observamos as cardinalidades máximas, nelas estarão presentes apenas os valores 1 e N, podendo haver três tipos diferentes de relacionamentos. Vejamos cada um deles, a seguir.

a. Relacionamento 1:1 (um-para-um)

O relacionamento 1:1 entre duas entidades A e B indica que uma ocorrência de **A** está associada a, no **máximo**, **uma** ocorrência de **B**, e uma ocorrência em **B** está associada a, no **máximo**, **uma** ocorrência em **A**.

Por exemplo, observe a seguinte situação:



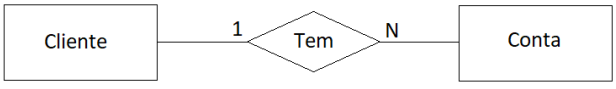
Neste caso, estamos indicando que um funcionário pode gerenciar apenas um departamento, e que um departamento deve ser gerenciado por um único funcionário.

Em um relacionamento deste tipo, há apenas uma chave estrangeira, em uma das entidades que indicar qual é o elemento da outra entidade, que está relacionado com ele. No exemplo acima, podemos ter um atributo chamado "gerente" na entidade Departamento, para esta situação.

b. Relacionamento 1:N (um-para-muitos)

Quando um relacionamento entre as entidades A e B tem por cardinalidade máxima 1 e N, indicamos que uma ocorrência de **A** está associada a várias ocorrências de **B**, porém uma ocorrência de **B** deve estar associada a, no **máximo**, **uma** ocorrência em **A.**

Vejamos um exemplo disso,



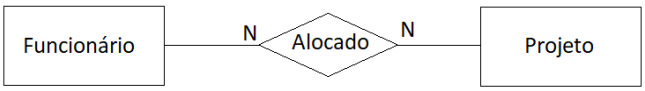
Estamos, aqui, indicando que um Cliente pode ter várias contas, mas uma conta deve pertencer a um único cliente, ou seja, não podemos ter “contas conjuntas” no nosso modelo.

Em um relacionamento deste tipo, há um único lugar para colocarmos o atributo de chave estrangeira. Ele deve estar no lado do indicador N. No caso, cada conta deve conter o atributo que indica quem é o seu proprietário.

c. Relacionamento N:N (muitos-para-muitos)

Finalmente, temos o relacionamento N para N entre duas entidades A e B. Neste caso, uma ocorrência de **A** está associada a **qualquer número** de ocorrências de **B**, e uma ocorrência em **B** está associada a **qualquer número** de ocorrências em **A**.

Vamos a um exemplo desta situação:



Estamos indicando que um funcionário pode estar alocado em vários projetos, ao mesmo tempo, e que um projeto pode ter vários funcionários trabalhando nele, de forma simultânea.

Quando um relacionamento é do tipo “muitos para muitos”, precisa-se, necessariamente, de uma relação ou tabela (no conceito do modelo relacional), para armazenar os dados dela. Assim, a relação deverá conter como atributos as chaves estrangeiras que indicam as instâncias, de onde ela se relaciona.

3.3.4 SÍMBOLOS DO MODELO ER

Observe a Figura 16 a seguir, a qual apresenta um resumo de todos os símbolos presentes em um modelo ER, com o significado de cada um deles.

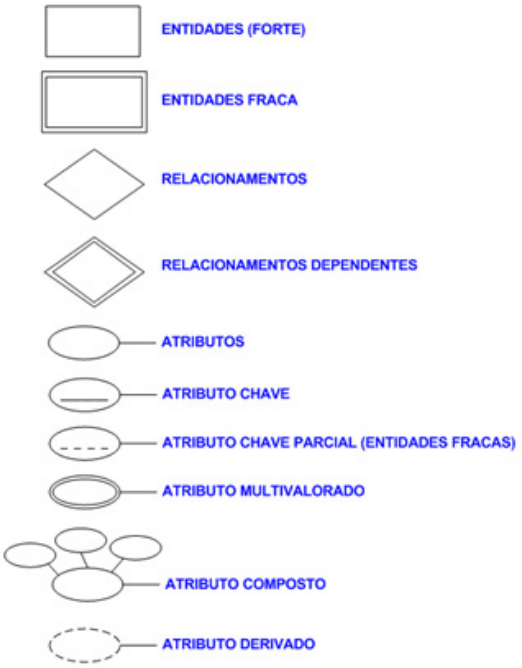


Figura 16 - Símbolos do Modelo ER.
Fonte: ELSMARI e NAVATHE (2019).

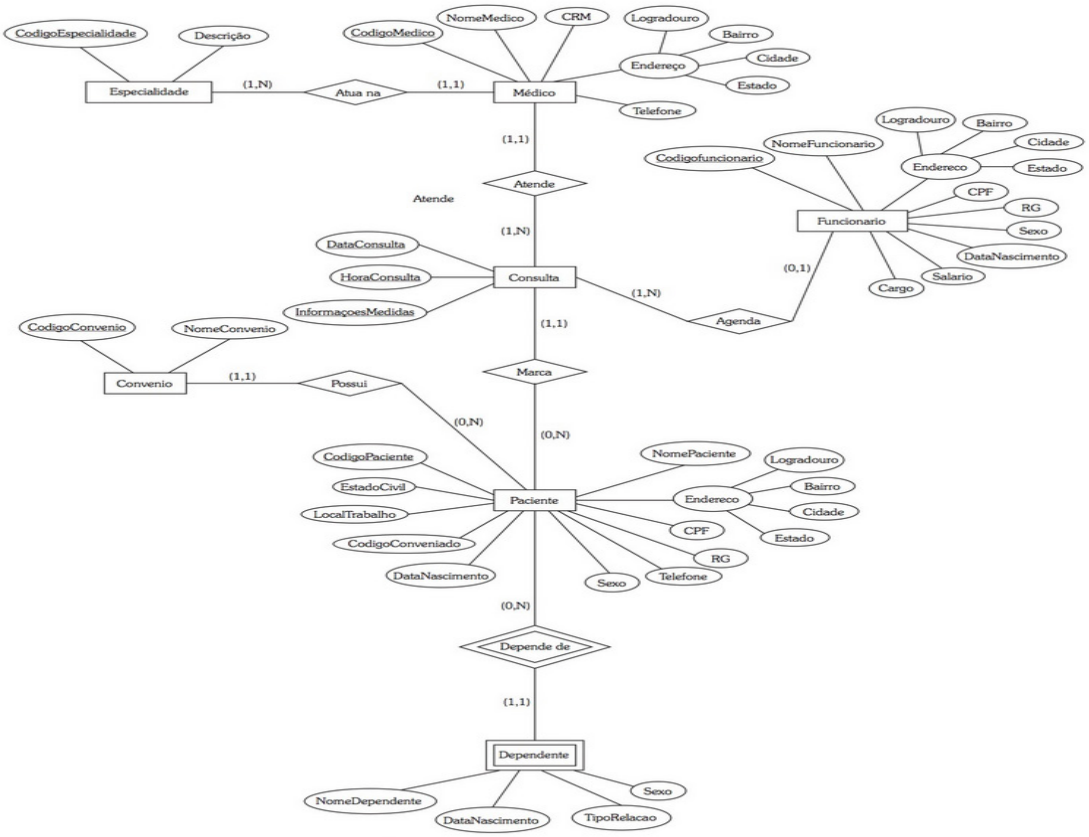


Figura 17 - Diagrama Entidade-Relacionamento.
Fonte: ALVES (2014).

O esquema conceitual criado, usando o Modelo Entidade-Relacionamento, chama-se Diagrama Entidade-Relacionamento (DER). Assim, MER é o conjunto de conceitos e elementos de modelagem que o projetista usa para criar o modelo. Já o DER é o resultado do processo de modelagem, ou seja, o seu produto final. Veja um exemplo de DER, na Figura 17 acima.

3.4 MODELO ER ESTENDIDO

O MER, apresentado na seção anterior, possui alguns problemas, quando trabalhamos com o paradigma da orientação a objetos. Ele é, por definição, orientado aos bancos de dados relacionais e à programação estruturada. Por esta razão, surgiu o Modelo Entidade-Relacionamento Estendido.

Um Modelo Estendido inclui todas as características presentes no MER tradicional e ainda traz conceitos adicionais para: subclasses e superclasses; especialização e generalização; categorias e tipos de união; e herança de atributos e relacionamentos. Estes são conceitos da Orientação a Objetos, muito aplicada, atualmente, no projeto e desenvolvimento de aplicações. Para os conceitos básicos relacionados a este paradigma, recorra aos seus cadernos de Programação I e II.

3.4.1 SUBCLASSE, SUPERCLASSE E HERANÇA

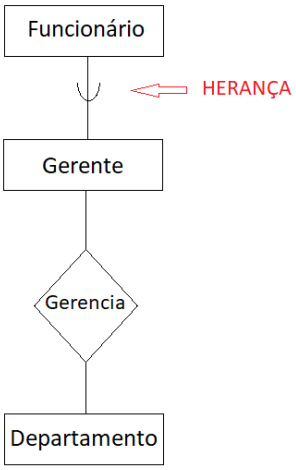
Em modelagem OO, representamos a superclasse e a subclasse, através do conceito de herança, em que uma classe derivada herda todas as características da sua classe pai. No ER Estendido, uma entidade pode ter subgrupos adicionais do seu tipo. Neste caso, uma entidade que é membro de uma subclasse herda todos os atributos e relacionamentos da entidade principal.

Considere, a título de exemplo, uma situação, em que necessitamos definir uma entidade, para representar os funcionários da empresa dentro do modelo. Por outro lado, podemos ter diferentes tipos de funcionários nesta empresa, elementos como: Gerente, Técnico, Engenheiro, Secretário, dentre outros. Como todos os elementos são derivados de Funcionário, eles necessitam de todas as funções que a entidade possa apresentar. Além disso, cada uma destas entidades derivadas pode ter atributos próprios.

Na extensão do MER, vamos representar esta situação com um subtipo ou subclasse

de uma entidade. Para tal, usamos um símbolo especial, na linha que liga as entidades, as quais estão representando a situação, o símbolo (semelhante à operação de união entre conjuntos).

Vamos a um exemplo desta representação. Suponha que a entidade Funcionário tenha uma subclasse para representar os gerentes e, estes, por sua vez, tenham um relacionamento com o departamento que gerenciam. Tal situação está representada no seguinte modelo:



Nessa situação, caso a entidade derivada tenha atributos próprios, basta indicá-los ao lado dela, como acontece no modelo normal. A implementação disso, na base de dados, será realizada, através de um elemento próprio, ou seja, através de uma relação ou tabela na base. Uma entidade não pode existir, no banco de dados, simplesmente por ser membro de uma subclasse; ela também precisa ser membro da superclasse. Assim, uma entidade pode ser incluída como membro de qualquer número de classes.

Outra informação relevante é que podemos ter herança simples ou múltipla. Na herança simples, uma entidade herda uma única outra entidade. Na herança múltipla, uma mesma subclasse herda mais de uma superclasse e, através disto, herda todas as características de todas as superclasses que foram herdadas por ela.

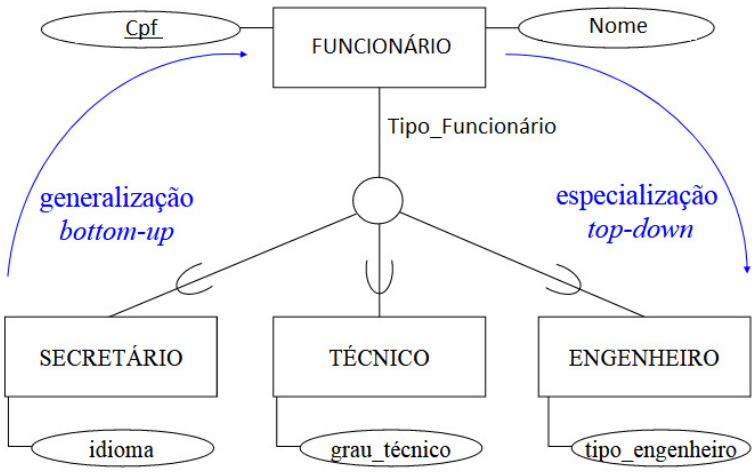
3.4.2 ESPECIALIZAÇÃO E GENERALIZAÇÃO

Especialização é um processo, através do qual definimos um conjunto de subclasses de uma mesma entidade. Este conjunto de subclasses é estabelecido, com base em algumas características distintas da entidade na superclasse.

No modelo estendido, as subclasses que definem a especialização são conectadas por

linhas a um círculo, representando a especialização, e este, por sua vez, está conectado à superclasse. Quando analisamos a situação, no sentido *bottom-up* (de baixo para cima), observamos uma generalização da entidade. Ao analisarmos no sentido contrário *top-down* (de cima para baixo), estamos especializando a entidade.

Como exemplo, suponha que tenhamos as derivações da entidade Funcionário, para representarmos os Engenheiros, Técnicos ou Secretários.



Observe que ainda usamos o símbolo de herança nas derivações, e que há um nome dado ao processo: "Tipo_Funcionário". Note, também, que existem atributos em cada uma das entidades derivadas. Isso implica que cada uma delas será representada, separadamente, no banco de dados. Além disso, teremos que usar um atributo "tipo_funcionário" na entidade Funcionário, para diferenciarmos os tipos derivados.

Neste caso, é importante compreender que as subclasses serão definidas pelo predicado ou atributo, mas também pela sua condição. Com relação à condição da derivação, podemos aplicar restrições, que serão apresentadas no próximo subitem.

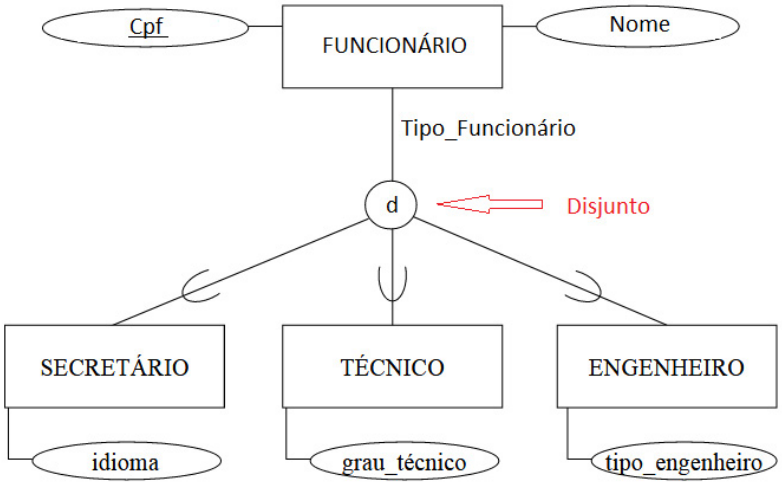
3.4.3 RESTRIÇÕES

Em um modelo, podemos ter várias situações, envolvendo especialização e generalização, mas nem todas são aplicadas de forma idêntica. Existem generalizações, nas quais podemos ter apenas uma das subclasses envolvidas, e ter outras, em que há a possibilidade de termos várias delas, de forma simultânea. Existem dois tipos de restrições que podemos aplicar ao modelo: restrições de disjunção e restrições de completude.

a. Restrições de Disjunção

Nesta situação, podemos ter subclasses que são mutuamente exclusivas, ou que se sobrepõem na representação. A primeira situação quer dizer que podemos ter apenas uma das subclasses sendo usadas para cada instância da superclasse, isto é, uma entidade de uma superclasse deve ser membro, quando muito, de apenas uma única subclasse. Isso é representado pelo mesmo símbolo, indicado antes, mas com uma letra "d" (de *disjoint* ou disjunto) dentro do círculo.

Por exemplo, vamos supor que um funcionário, dentro do modelo de negócio que estamos representando, possa ser secretário, técnico ou engenheiro, mas não mais do que um destes ao mesmo tempo. Assim, o modelo deveria indicar:



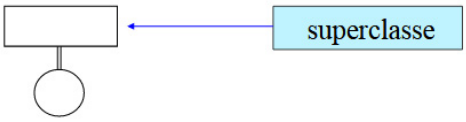
Também podemos ter a situação, na qual as subclasses podem acontecer de forma simultânea, ou seja, elas se sobrepõem. Neste caso, uma entidade de superclasse pode ser membro de mais de uma subclasse, devendo-se usar, em vez da letra "d", a letra "o" (de *overlap* ou sobreposição).

Em nosso exemplo acima, isso indicaria que um Funcionário poderia ser, ao mesmo tempo, Secretário e Técnico ou Engenheiro, ou todos, simultaneamente.

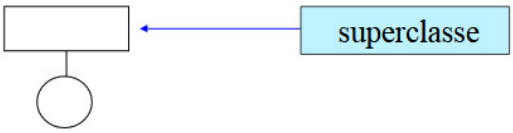
b. Restrições de Completude

Em relação às restrições de completude, podemos indicar que cada entidade de uma superclasse deve ser membro de alguma subclasse na especialização, ou isto é opcional.

Assim, podemos ter a representação da Restrição de Totalidade na generalização. Ela indica a obrigatoriedade mencionada acima, representado no modelo, através da seguinte situação:

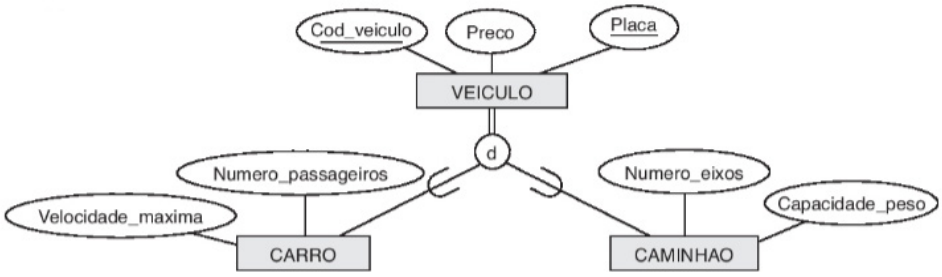


Observe que foi usada uma linha dupla, ligando a superclasse com o círculo, onde estará representada a especialização/generalização. Para indicarmos um processo opcional, cuja superclasse pode ou não ter uma das representações das subclasses, utiliza-se a ligação normal:



Usamos esta situação no modelo dos Funcionários, apresentado acima. Ali está indicado que um dado Funcionário pode ou não ser Gerente, Secretário, Técnico ou Engenheiro, sendo opcional àquela entidade.

Vamos interpretar um segundo modelo, exemplo em que estas situações de restrição estão presentes, obtido em ELSMARI, 2019. Veja:



Observando com atenção, verificamos que há uma superclasse, definida pela Entidade Veículo, havendo duas possíveis especializações dela, como Carro e Caminhão. Cada uma das entidades derivadas tem os seus atributos próprios: carro tem velocidade_maxima e numero_passageiros; já caminhão tem numero_eixos e capacidade_peso. Por outro lado, ambas as entidades herdam os atributos cod_veiculo, preço e placa, vindos da entidade Veículo.

Podemos observar, ainda, que há uma especialização/generalizada do tipo disjunta (veja o "d" no círculo). Neste caso, estamos informando que um veículo pode ser um carro ou um caminhão, mas não os dois ao mesmo tempo. Outra informação importante vem do tipo de especialização que foi realizada (restrição de completude). Ali, estamos indicando

a obrigatoriedade de especialização, ou seja, um Veículo deve, obrigatoriamente, ser classificado como Carro ou Caminhão; não é opcional.

3.4.4 DIAGRAMA ESTENDIDO

Assim como o modelo estendido é o conjunto de conceitos, apresentados anteriormente, o diagrama estendido é a representação completa destes conceitos. O Diagrama EER é o resultado final do processo de modelagem, feito pelo projetista e, normalmente, entregue à equipe responsável pelos demais processos de modelagem e implementação da base de dados.

A Figura 18, a seguir, mostra um diagrama EER completo, obtido em ELSMARI, 2019. Este modelo possui vários dos recursos apresentados aqui, para modelar um banco de dados de uma Universidade. Em especial, observe a entidade ALUNO_COLABORADOR, na qual há a herança múltipla, pois herda, simultaneamente, as entidades FUNCIONÁRIO e ALUNO.

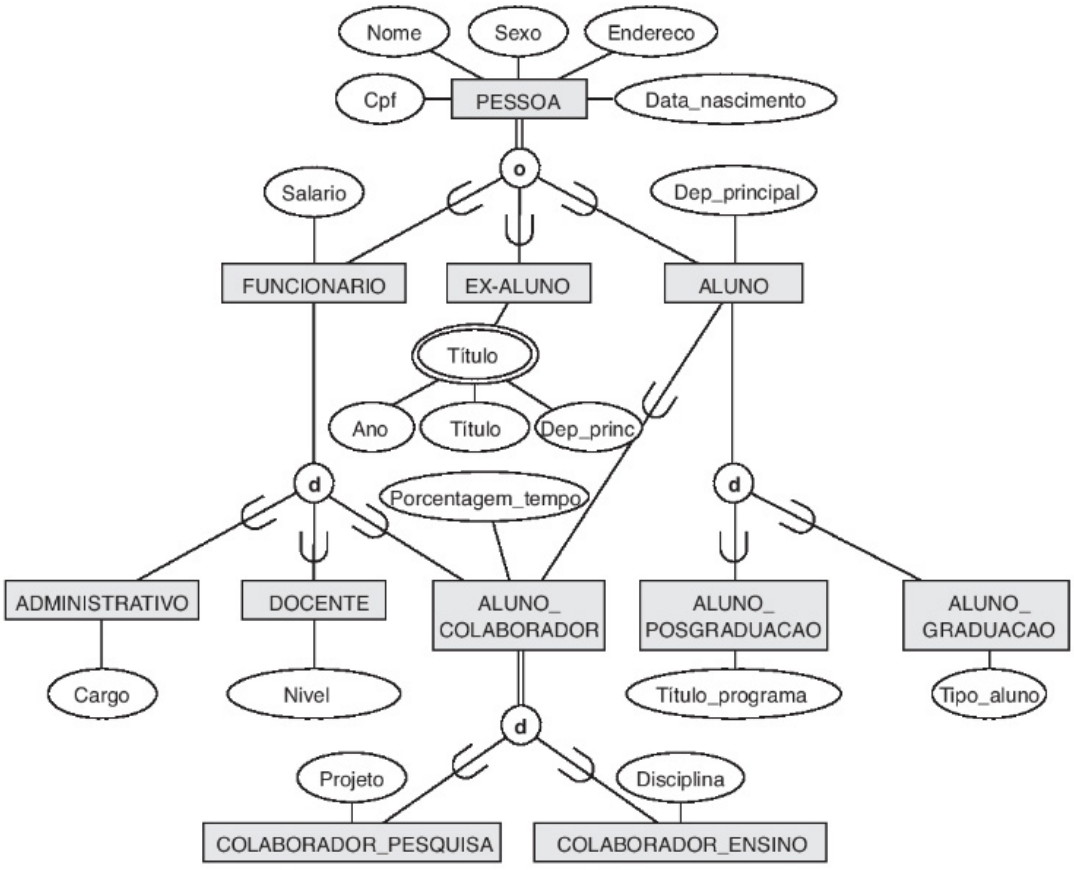


Figura 18 - Banco de dados de uma Universidade.
Fonte: ELSMARI, 2019 (pag. 168).

3.5 MODELAGEM COM UML

Atualmente, é muito comum o uso da Linguagem UML para a modelagem de sistema e, dentro desta linguagem, podemos usar o Diagrama de Classe como um recurso para modelamos o banco de dados. Para maiores informações desta linguagem, verifique SOMERVILLE, 2011.

Na modelagem OO, cada objeto será representado por uma classe (sim, este é um conceito de programação, mas será usado aqui), utilizada para representar as nossas relações ou entidades no banco de dados. Assim, os atributos da classe correspondem aos atributos da relação no banco de dados. Nos dois casos, o conceito de instância é o mesmo.

Por outro lado, em uma modelagem UML, os métodos de cada classe não interessam à modelagem do banco, apenas para a implementação posterior do sistema, que fará uso dos dados.

Outro elemento que não está presente é o relacionamento N:N. Em um diagrama UML, cada relacionamento é representado por uma classe diferente, contendo as chaves estrangeiras que fazem parte da relação. Os demais relacionamentos 1:1 e 1:N podem ser representados diretamente com as chaves estrangeiras dentro da classe interessada.

Em um diagrama de classe, existem elementos presentes na modelagem do banco de dados (na verdade, estes derivam da modelagem OO), como herança, agregação e composição. No diagrama de classes, também há a noção de multiplicidade da relação, a qual podemos usar, quase que diretamente, para o conceito de cardinalidade no banco de dados. Na Figura 19, estão apresentadas as possíveis relações entre classes de um modelo.

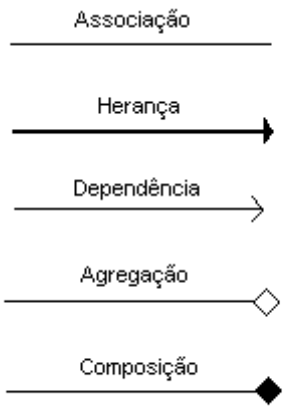
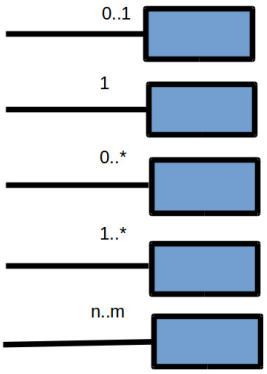
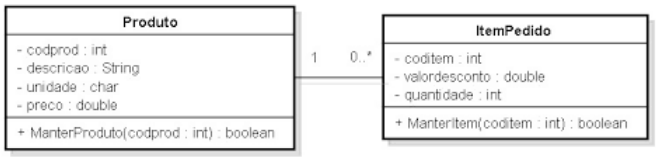


Figura 19: Relações no diagrama de classes.
Elaborado pelo autor (Junho de 2020).

Ao lado de cada uma das relações, podemos indicar a cardinalidade dela. Para isto, indicamos uma das seguintes medidas:

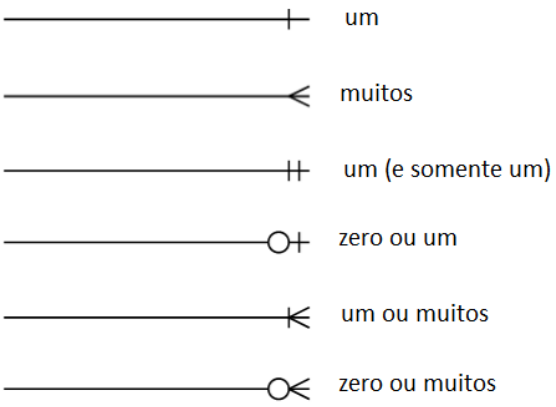


Nesta notação, o símbolo * (asterisco) pode ser usado para representar a cardinalidade "várias", assim como o n..m. A multiplicidade 1 (um) é usada para a associação 1:1. Outro detalhe relevante é que devemos sempre observar, no sentido de uma classe A para a classe B, esta relação tem sua multiplicidade indicada ao lado da classe B. Observe:



No exemplo, podemos observar que um determinado produto não está obrigado a participar de um pedido, mas pode estar a vários (0..*). Já quando olhamos, do ItemPedido para o produto, percebemos que, para existir o pedido, é obrigatório que um produto participe dele.

Por mais que esta seja a notação mais comum, podemos encontrar outras formas de representar a cardinalidade nas associações entre as classes. Quando isso acontece, os símbolos usados nas ligações são os seguintes:



Assim sendo, um diagrama de classe, com relativamente pouco trabalho, pode ser usado (e considerado) no processo de modelagem conceitual do banco de dados. Isso evita um processo adicional, uma vez que o usuário final tem mais dificuldade de compreender e auxiliar na modelagem, ao usarmos este processo. Veja um diagrama de classe na Figura 20. Faça uma análise dele e busque identificar as entidades do banco, além dos relacionamentos que serão necessários em sua implementação.

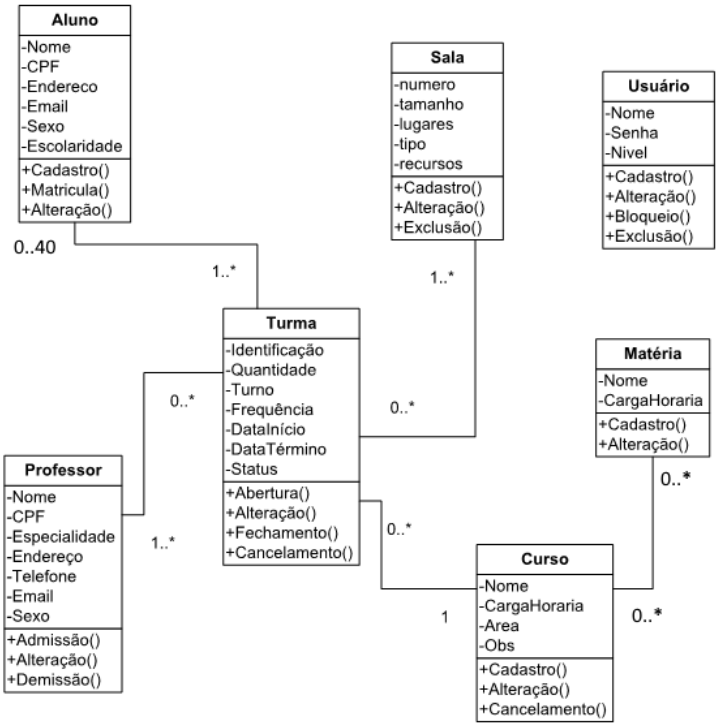
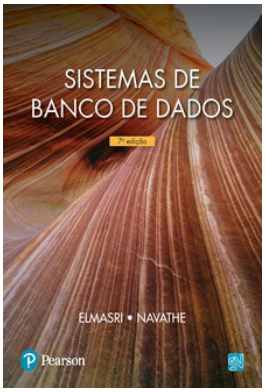


Figura 20 - Diagrama de classe, representando um pequeno controle de uma escola de cursos e treinamentos.
Fonte: https://www.luis.blog.br/orientacao-a-objetos-classe-e-objeto-propriedades-e-metodos.html. (Acesso em: junho/2020).

SAIBA MAIS

Busque maiores informações sobre o processo de modelagem conceitual do banco de dados, na série de vídeos da Bóson Treinamentos. O primeiro deles pode ser acessado em: https://www.youtube.com/watch?v=W2Z1STnjNJo. Não deixe de olhar os demais vídeos, a respeito do assunto.

SUGESTÃO DE LIVRO



O livro Sistema de Banco de Dados, de autoria de Ramez Elmasri e Shamkant Navathe, contém muitas informações sobre a modelagem conceitual. Ele foi utilizado com frequência nesta unidade. Em especial, você pode ler os capítulos sobre Modelo Entidade-Relacionamento e Modelo Entidade-Relacionamento Estendido, para se aprofundar nos conhecimentos adquiridos.

CONSIDERAÇÕES FINAIS

Realizar a modelagem conceitual do banco de dados é fundamental, para que a construção de um banco de dados consiga responder, de forma adequada, às demandas dos processos de software que fazem uso deles. Nesta unidade, foram apresentadas as técnicas de modelagem conceitual, aplicadas aos SGBDs.

A técnica de uso mais comum baseia-se no modelo ER, de Peter Chen, o qual, mesmo tendo mais de 40 anos, representa uma importante metodologia no projeto conceitual dos BD. Através dela, descrevemos cada uma das entidades que irão compor o banco de dados, seus atributos, e como elas se relacionam entre si.

Além de ser um modelo completo e funcional, a MED tem um grande atrativo na facilidade de compreensão dos seus conceitos pelos leigos, o que a torna ideal para apresentar a ideia de modelagem aos usuários finais, esperando-se contribuições dele no modelo.

Com os processos de construção de software, baseados na orientação a objetos, a proposta de Chen sofreu adaptações e melhorias, ao se criar o Modelo Entidade-Relacionamento Estendido, contemplando, também, conceitos que abrangem a herança, super e subclasses e restrições, durante o processo. Isto tornou o modelo ER mais próximo da realidade de implementação.

Na terceira unidade, também foram apresentados, de forma introdutória, alguns conceitos da Linguagem UML que podem ser usados no processo de modelagem conceitual, especialmente o Diagrama de Classes.

EXERCÍCIO FINAL

1. Sobre cardinalidade na modelagem entidade-relacionamento, considere:

I- No relacionamento 1:1, cada entidade só pode se relacionar com uma entidade do outro conjunto. No relacionamento 1:N, cada entidade do primeiro conjunto pode se relacionar com qualquer entidade do segundo conjunto, mas as entidades do segundo conjunto não podem se relacionar com várias entidades do primeiro conjunto.

II- Na notação de par ordenado, (0,1):(1,N) o primeiro número do par indica a cardinalidade mínima e o segundo, a máxima. A cardinalidade mínima indica uma exigência da participação de uma instância da entidade em relacionamentos.

III- No relacionamento N:M, qualquer número de relacionamentos entre as entidades

é válido, podendo ser implementado no banco, sem a necessidade de relações auxiliares.

É correto o que consta em:

a) Apenas I.

b) I, II e III.

c) Apenas I e II.

d) Apenas I e III.

e) Apenas II.

2. Considere a seguinte estrutura:

ALUNO (cpf : string , nome : string , endereco : string, telefone : string)

MATRÍCULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number)

A representação do esquema relacional acima, segundo um diagrama de entidades e relacionamentos, deve representar ALUNO, MATRÍCULA e CADEIRA, respectivamente, como:

a) entidade, relacionamento 1xn e entidade.

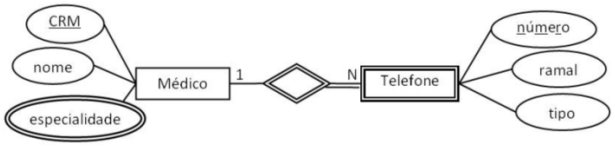
b) entidade, entidade e atributo.

c) entidade, relacionamento nxm e entidade.

d) entidade, entidade e relacionamento nxm.

e) relacionamento 1xn, entidade e entidade.

3. Considere o diagrama ER (Entidade-Relacionamento), a seguir, sobre Médicos, suas especialidades e vários telefones.



Neste diagrama, as entidades são retângulos, os relacionamentos são losangos, os atributos são ovais, os atributos multivalorados são ovais, com linhas duplas, as entidades fracas são retângulos, com linhas duplas, e os relacionamentos identificadores são losangos, com linhas duplas. O diagrama precisa ser mapeado, a fim de armazenar dados em um Sistema de Gerenciamento de Bancos de Dados Relacional. Com base nas regras de mapeamento e da Terceira Forma Normal (3FN), atribua V (verdadeiro) ou F (falso) às afirmativas a seguir.

( ) A chave primária da tabela Telefone será composta por CRM e número.

( ) Uma tabela específica será criada para Médico, e outra para Telefone, com uma chave estrangeira.

( ) Uma tabela específica será criada para Médico, e outra para Telefone, sem qualquer chave estrangeira.

( ) Uma tabela específica será criada para o atributo Especialidade, com uma chave estrangeira para a tabela Médico.

( ) Uma tabela específica será criada para o relacionamento entre Médico e Telefone, com as respectivas chaves estrangeiras.

Assinale a alternativa que contém, de cima para baixo, a sequência correta.

a) V, V, F, V, F.

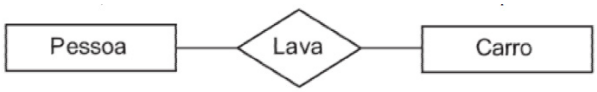
b) V, F, V, F, V.

c) F, V, F, V, F.

d) F, V, F, F, V.

e) F, F, V, V, V.

4. Considere o modelo entidade-relacionamento, representado abaixo.



Na transformação deste modelo conceitual Entidade-Relacionamento em um modelo lógico relacional, as cardinalidades do relacionamento entre as entidades exercem papel importante. Dado que se deseja gerar um modelo relacional correto, pode-se afirmar que sempre darão origem a uma tabela, para cada uma das entidades relacionadas, os relacionamentos do tipo:

a) (0,n) x (0,n), podendo ou não gerar uma tabela para o relacionamento.

b) (0,1) x (0,n), podendo ou não gerar uma tabela para o relacionamento.

c) (0,1) x (1,1), gerando uma tabela para o relacionamento.

d) (1,n) x (1,n), devendo gerar uma tabela para o relacionamento.

e) (1,1) x (1,n), devendo gerar uma tabela para o relacionamento

5. Um mercado que comercializa alimentos hortifrutigranjeiros faz compras diárias de diversas fazendas e enfrenta prejuízos, decorrentes da falta de controle relacionada ao prazo de validade de seus produtos. Para resolver este problema, o proprietário resolveu investir em informatização, que proporcionará o controle do prazo de validade,

a partir da data da compra do produto. A modelagem de dados proposta pelo profissional contratado apresenta três tabelas, ilustradas a seguir, sendo que o atributo Código, nas tabelas Produto e Fazenda, é unívoco.

Produto
Código
Tipo de Produto
Descrição

Estoque
Produto
Fazenda
Data da compra
Validade do Produto
Quantidade

Fazenda
Código
Nome
Endereço
Telefone

A partir das informações acima, é correto concluir que:

a) o relacionamento entre as tabelas Produto e Estoque é do tipo um-para-muitos.

b) o campo Produto, na tabela Estoque, não pode fazer parte da chave, e corresponde ao campo Descrição na tabela Produto.

c) o campo Fazenda, na tabela Estoque, deverá ser a chave primária nesta tabela e corresponde ao campo Código na tabela Fazenda.

d) o campo Código é chave primária, na tabela Produto, e identifica a fazenda fornecedora do produto.

e) a tupla {produto, fazenda} deverá ser usada como a chave primária da tabela Estoque.

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados**. 1. ed. São Paulo: Érica, 2014.

AMADEU, Cláudia (org). **Banco de Dados**. São Paulo: Pearson, 2014.

BARBOZA, Fabrício. **Modelagem e Desenvolvimento de Banco de Dados**. Porto Alegre: SAGAH, 2018.

CARDOSO, Virginia. CARDOSO, Giselle. **Sistemas de Banco de Dados: uma abordagem introdutória e prática**. São Paulo: Saraiva, 2012.

CARDOSO, Virginia; CARDOSO, Giselle. **Linguagem SQL**: **fundamentos e práticas**. São Paulo: Saraiva, 2013.

DATE, Chris. Introdução a Sistemas de Banco de Dados. 8.ed. São Paulo, Campus: 2004.

ELSMARI, Ramez. NAVATHE, Shamkant**. Sistemas de Banco de Dados**. 7. ed. São Paulo: Pearson, 2019.

HEUSER, Carlos. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

MACHADO, Felipe. Banco de Dados: Projeto e Implementação. 4. ed. São Paulo: Érica, 2014.

MEDEIROS, Luciano. **Banco de dados: princípios e práticas**. Curitiba: Intersaberes, 2013.

MANNINO, Michael V. **Projeto, desenvolvimento de aplicações e administração de banco de dados**. 3. ed. Porto Alegre: AMGH, 2014.

RAMARKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados**. 3. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de Software**, 9. Ed. São Paulo: Pearson Prentice-Hall, 2011.

TEOREY, Toby et. al. **Projeto e Modelagem de Banco de Dados**. 2. ed. São Paulo: Elsevier, 2014.

unidade 4

LINGUAGEM SQL

4.1 INTRODUÇÃO À UNIDADE

Figura 21 - Linguagem SQL.
Fonte: <https://www.tecmundo.com.br/software/146482-sql-que-ele-serve.htm> Acesso em: Julho de 2020.

Bem-vindos à ultima unidade deste caderno!

Ao falarmos em usar dados em uma base relacional, não há como ignorar o uso da linguagem mais popular (em alguns casos, a única!), para manipulação de informações neste tipo de SGBD. Assim, é essencial que a Linguagem SQL seja conhecida por todos aqueles que desenvolverão sistemas e usarão este tipo de base de dados.

Na Unidade 4, vamos conhecer as características desta linguagem, seus princípios fundamentais e seus elementos de construção. Neste texto, abordaremos apenas a Linguagem SQL, e não o seu uso, através de linguagens de programação, visto que, em cada uma delas existem recursos específicos para enviar os comandos em SQL ao banco e tratar o seu retorno. Esta tarefa fica a cargo de cada um dos leitores, depois de decidir qual é a linguagem-mãe que lhe interessará. Existem vastas informações em como fazer isso, com linguagens como Java, Php e outras.

A princípio, conheceremos alguns aspectos básicos da Álgebra Relacional que fornece os fundamentos matemáticos para a construção da SQL. Em uma visão mais ampla e prática, SQL fornece comandos, para a execução das operações da álgebra relacional.

Após isso, veremos como o SQL é estruturado e quais são os seus elementos principais. Neste ponto, analisaremos separadamente as sublinguagens de manipulação e de consulta aos dados. Nossa unidade será encerrada com a explanação do conceito de visão e seu uso na base de dados, um conceito fundamental para garantirmos uma maior segurança no acesso aos nossos dados.

Os principais objetivos de aprendizagem desta unidade são: conhecer o processo de

modelagem de banco de dados e sua transformação em um modelo físico; transformar elementos da análise de requisitos em entidades de um banco relacional; compreender os diferentes relacionamentos entre as entidades, bem como apresentá-los no processo de modelagem; definir um esquema de banco de dados derivado do processo de modelagem; conhecer e aplicar os conceitos das operações da álgebra relacional e sua transcrição para comandos da Linguagem SQL; aplicar a Linguagem SQL nas operações de manipulação de bases de dados relacionais.

4.2 ÁLGEBRA RELACIONAL: CONCEITOS BÁSICOS

De uma forma simplista, o que é "álgebra"? Podemos entender que este conceito diz respeito a apresentarmos uma definição formal e sustentada a um conjunto de operadores e das operações que são permitidas sobre eles.

Seguindo a mesma abordagem, podemos dizer que a álgebra relacional é a definição de um conjunto de operadores, e as operações que são possíveis sobre eles, bem como a formação do resultado final destas operações.

4.2.1 CONCEITOS INICIAIS

A álgebra relacional tem como operadores as relações. Por relação, entendemos o conjunto dos nossos dados, exatamente como foi apresentado na definição das bases relacionais na Unidade I (aqui há uma inversão feita no material, o conceito de relação da base de dados é originário do conceito da álgebra relacional, e não o contrário!).

Empregado

codEmp	Nome	Salario	idade	codDep
200	Pedro	3.000,00	45	001
201	Paulo	2.200,00	43	001
202	Maria	2.500,00	38	001
203	Ana	1.800,00	25	002

Projeto

codProj	Descricao	codDep
A	AATOM	001
B	DW espaço-temporal	002

ProjetoEmpregado

codProj	codEmp	dataIn	dataFi
A	200	01/01/2007	atual
A	201	01/01/2007	atual
A	202	01/02/2006	18/02/2010
B	203	15/02/2008	15/02/2010

Departamento

codDep	descricao
001	Pesquisa
002	Desenvolvimento

Figura 22 - Exemplos de relações.
Fonte: Elaborado pelo autor (Julho/2020).

Cada relação é formada por linhas, que são chamadas de tuplas, e, cada uma das tuplas é separada por atributos que são os seus elementos constituintes. Por exemplo, observe a Figura 22, acima, na qual algumas relações são apresentadas. Nossos exemplos de aplicação da Álgebra Relacional serão feitos sobre estes exemplos. No conjunto de relações, podemos perceber que existem chaves primárias e estrangeiras, as quais foram colocadas em cores diferentes, embora isto não seja relevante para a compreensão da álgebra relacional.

A álgebra relacional é uma linguagem formal de manipulação de dados, orientada a conjuntos (que são as relações), sendo fechada, quanto aos seus resultados. Isso quer dizer que o resultado de qualquer uma das suas operações deve ser expresso em uma relação com as mesmas características presentes nos seus operadores. Assim, temos sempre a garantia de que o resultado de uma operação qualquer pode sempre servir como entrada para uma próxima operação, sem prejuízo da sua execução. Podemos dizer que os operadores da álgebra relacional recebem uma ou mais relações com entrada e geram uma nova relação na saída, permitindo-se que sejam realizadas consultas e alterações nas relações.

Existem motivos claros para o estudo e a compreensão desta álgebra. A principal delas é que, compreendendo a álgebra relacional, facilitará apreender a linguagem SQL. Não existe nenhum SGBD que implementa a álgebra em sua linguagem de manipulação, mas elas incorporam os seus conceitos, como no caso da SQL. Finalmente, caso tenhamos a necessidade de otimizar algoritmos de consulta às bases de dados relacionais, precisamos conhecer e aplicar elementos da álgebra relacional.

Os operadores da álgebra relacional são divididos em dois grupos: unários e binários. Os operadores unários operam sobre uma única relação, enquanto os binários operam sobre duas relações. Além destes, veremos outros dois grupos que são os operadores derivados e os especiais. Vamos a eles.

4.2.2 OPERADORES UNÁRIOS

Os operadores unários são os operadores da álgebra relacional que atuam sobre uma única relação de entrada, produzindo outra relação na sua saída.

a) Seleção (σ)

Este operador atua sobre uma relação e produz como resultado um conjunto de tuplas da relação original, que atendem a algum critério de filtragem. É importante salientar que o resultado da sua aplicação deve conter os mesmos atributos da relação de entrada. Sua sintaxe é dada por:

$$\sigma_{\text{<condição de seleção>}}(\text{<R>})$$

Nela, o símbolo σ (sigma) representa a operação de seleção. Já <condição de seleção> é uma expressão booleana, a qual envolve elementos literais e atributos da relação de entrada R. Na condição de seleção, podemos usar qualquer um dos atributos da relação R, bem como operadores de comparação (: =, <, <=, >, >=, <>), ou relacionais (*and*, *or* e *not*).

Por exemplo, usando as relações da Figura 19, podemos selecionar os empregados que tenham salário menor do que R$ 2.000,00. Vejamos:

$\sigma_{\text{salario} < 2000}(\text{Empregado})$

Empregado

codEmp	Nome	Salario	idade	codDep
200	Pedro	3.000,00	45	001
201	Paulo	2.200,00	43	001
202	Maria	2.500,00	38	001
203	Ana	1.800,00	25	002

Resultado

codEmp	Nome	Salario	idade	codDep
203	Ana	1.800,00	25	002

Seguindo a mesma lógica, podemos buscar os dados dos empregados com salário maior que 2000, e com menos 45 anos, definindo:

$$\sigma_{\text{salario>2000 AND idade < 45}} (\text{Empregado})$$

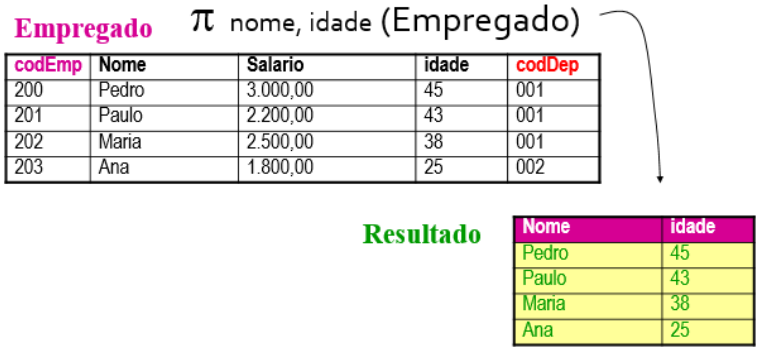
b) Projeção (π)

Esta operação retorna um ou mais atributos da relação de interesse. Seu resultado é uma relação, contendo apenas as colunas da relação original que foram previamente selecionadas. Sua sintaxe é:

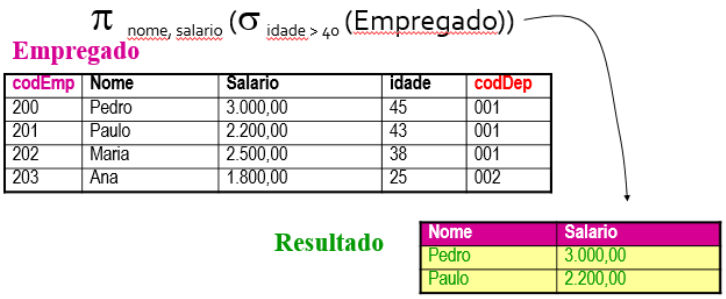
$$\pi_{\text{<lista de atributos>}} (\text{<R>})$$

Onde <lista de atributos> é uma lista que contém os nomes das colunas da relação <R> que foi dada como entrada da operação.

Como exemplo, podemos fazer uma projeção sobre a relação empregado, que resulta apenas nos atributos nome e idade, de todos eles:



É importante salientar que podemos juntar mais de uma operação em uma expressão. Devemos lembrar que o resultado de uma operação é uma relação e pode, por isso, ser usado como entrada para outra operação. Por exemplo, podemos buscar o nome e o salário dos empregados, que tenham mais de 40 anos:



Observe que foi realizada uma seleção dos empregados, com mais de 40 anos, e sobre ela foi aplicada a projeção que resultou apenas no nome e salário deles. Por outro lado, haveria problema se invertêssemos esta sequência. Caso tivéssemos projetado o nome e o salário de todos os empregados, não teríamos como fazer a seleção sobre o resultado, pois nele não haveria o atributo (coluna) idade. Se ela fosse incluída, a inversão seria possível e correta.

4.2.3 OPERADORES BINÁRIOS

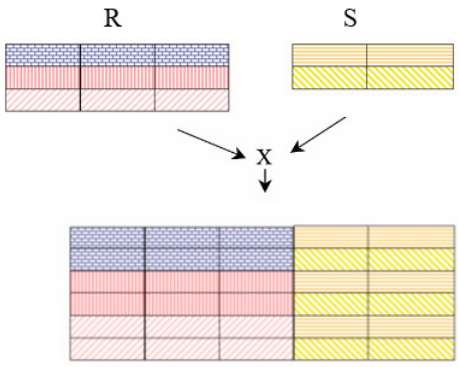
Os operadores binários atuam sobre duas relações fornecidas na sua entrada e resultam, na saída, em uma única e nova relação. As relações fornecidas na entrada devem ser compatíveis, ou seja, devem possuir o mesmo número de atributos, e o domínio de cada uma das colunas entre elas deve ser idêntico. Assim, o domínio da primeira coluna de A deve ser equivalente ao domínio da primeira coluna de B, e assim por diante.

a) Produto Cartesiano (x)

Esta operação, quando aplicada sobre duas relações R e S, retorna todas as combinações possíveis entre as tuplas das duas relações. O resultado é o mesmo da operação de nome igual, na álgebra de conjuntos da matemática, sendo uma nova relação de tuplas, onde cada uma das tuplas de R é concatenada (unida) a cada uma das tuplas de S. Sua sintaxe é:

R x S

Observe isso, demonstrado graficamente, a seguir. Na relação R existem três tuplas distintas (com cores diferentes). O mesmo acontece com a relação S. No resultado, cada uma das tuplas de R é juntada a cada uma das tuplas de S:



Note que o resultado terá a mesma quantidade de tuplas que o produto do número de tuplas das relações de entrada. Se a primeira relação de entrada tem 3 tuplas, e a segunda tem outras 2, o resultado terá 3x2 tuplas, como acima.

Vamos a um exemplo prático, usando as relações da Figura 19. Podemos fazer o produto cartesiano entre as relações Projeto e Departamento. Sua escrita e o resultado final serão:

Projeto x Departamento

Projeto			Departamento	
CodProj	Descrição	CodDep	CodDep	Descrição
A	AATOM	001	001	Pesquisa
A	AATOM	001	002	Desenvolvimento
B	DW espaço-temporal	002	001	Pesquisa
B	DW espaço-temporal	002	002	Desenvolvimento

Veja que o resultado, por mais que exista um relacionamento entra as duas relações originais (feito pela chave estrangeira CodDep em Projeto e a chave primária de mesmo nome em Departamento), não é considerado no produto cartesiano. O resultado junta todo mundo. Caso desejemos ter como resultado apenas os projetos e seus departamento relacionados, devemos fazer uma seleção, após o produto cartesiano:

$$\sigma_{\text{Projeto.codDep = Departamento.codDep}} \text{(Projeto x Departamento)}$$

b) União (U)

A operação de União é a mesma, realizada entre conjuntos na matemática tradicional. Requer que as duas relações fornecidas como argumento tenham o mesmo esquema e

resultem em uma nova relação, com o mesmo esquema, cujo conjunto de linhas é a união dos conjuntos de linhas das relações dadas como argumento. Ou seja, esta operação retorna a união das tuplas de R e S, eliminando, automaticamente, as tuplas duplicadas. Sua sintaxe é:

R U S

Observe que não poderíamos fazer uma união entre as relações Projeto e Departamento de forma direta, pois elas têm atributos com domínios diferentes. Para isso, deveríamos fazer, em primeiro lugar, uma projeção que organizasse. Então, a união seria possível.

$$\pi_{\text{codDep, Descrição}} \text{ (Projeto) U } \pi_{\text{codDep, Descrição}} \text{ (Departamento)}$$

Com isso, o resultado seria:

CodDep	Descrição
001	AATOM
002	DW espaço-temporal envolvimento
001	Pesquisa
002	Desenvolvimento

c) Intersecção (∩)

A operação de interseção é análoga a de conjuntos da matemática. Ela também tem as mesmas restrições da União, ou seja, os atributos e domínios devem ser iguais nas duas relações. O resultado desta operação é constituído apenas das tuplas que são idênticas nas duas relações (elementos repetidos apenas). Sua sintaxe é:

R ∩ S

Caso aplicássemos esta operação nas relações Projeto e Departamento, como feito acima, teríamos um resultado vazio, já que não têm tuplas iguais de código e descrição entre as duas relações. Caso existissem, poderíamos usar a operação, para determinar quais são os projetos que têm descrição igual àquela dos seus departamentos.

d) Diferença (-)

A operação de diferença resulta nas tuplas da primeira relação, que não tem tuplas idênticas na segunda relação. Desta forma, seu resultado elimina as duplicidades (resultado da interseção!), deixando aquilo que é exclusivo da primeira relação. Sua sintaxe é:

R - S

É importante salientar que esta operação não tem a propriedade comutativa com a união e a interseção. Desta forma, R – S tem resultado diferente de S – R, pois a ordem das relações é fundamental.

Poderíamos, como exemplo, usar a operação, para determinar quais são os empregados, do nosso modelo exemplo, que não estão alocados em nenhum projeto, escrevendo:

$$\pi_{codEmp}(Empregado) - \pi_{codEmp}(ProjetoEmpregado)$$

4.2.4 OPERADORES DERIVADOS

Estes operadores são chamados de derivados, uma vez que podem ser implementados com o uso das operações unárias e binárias, estudadas anteriormente. Aqui, temos dois novos operadores, que são a junção e a divisão.

a) Junção (|x|)

A operação de junção é o resultado de uma seleção, realizada dentro de um produto cartesiano, cujo resultado respeita a condição expressa na junção. Ela existe, apenas para que não seja necessário escrever as duas outras operações. Sua sintaxe é:

R |x|_{<condição>} S

Embora o resultado seja o mesmo, como mostra a figura a seguir, em termos de implementação em uma linguagem (com o SQL, por exemplo), a junção é mais eficiente do que fazer a seleção de um produto cartesiano:

$$R \bowtie_{cond} S = \sigma_{cond}(R \times S)$$

Usando o exemplo anterior, as duas operações a seguir teriam o mesmo resultado final:

$$\sigma_{\text{Projeto.codDep = Departamento.codDep}} (\text{Projeto x Departamento})$$

$$\text{Projeto } |x|_{\text{Projeto.codDep = Departamento.codDep}} \text{ Departamento}$$

SAIBA MAIS

Na álgebra relacional completa, existem diferentes tipos de junção (assim como no SQL) que são as junções à esquerda (*left join*), à direita (*right join*), junção completa (*full join*), dentre outras. Para maiores informações sobre elas, busque o livro Sistemas de Banco de Dados, de Elsmari e Navathe, o qual está nas referências desta unidade.

b) Divisão (÷ ou /)

A operação de divisão usa duas relações R e S, onde a relação R tem mais atributos que a relação S, mas dentre os seus atributos devem estar aqueles da relação S. O resultado será formado pelos atributos de R, nos quais os atributos de S aparecem com o mesmo conteúdo. Sua sintaxe é:

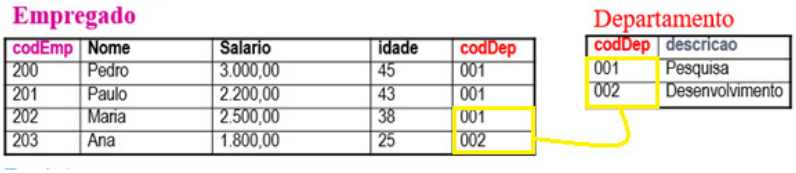
$$\mathbf{R \div S}$$

Por exemplo, vamos considerar uma divisão, realizada entre o nome e o código de departamento dos Empregados e os códigos de departamento da relação Departamento, com a seguinte expressão:

$$\pi_{\text{nome, codDep}} (\text{Empregado}) \div \pi_{\text{codDep}} (\text{Departamento})$$

Neste caso, teremos como resposta apenas os empregados Maria e Ana, pois apenas

nelas os códigos de departamento aparecem como na relação Departamento (001 e 002, nesta ordem):



Empregado

codEmp	Nome	Salario	idade	codDep
200	Pedro	3.000,00	45	001
201	Paulo	2.200,00	43	001
202	Maria	2.500,00	38	001
203	Ana	1.800,00	25	002

Departamento

codDep	descricao
001	Pesquisa
002	Desenvolvimento

4.2.5 OPERADORES ESPECIAIS

Finalmente, existem dois outros operadores, considerados como especiais, nos quais podemos mudar o nome de uma relação e atribuir o conteúdo de uma relação para outra. São os operadores de renomeação e atribuição.

a) Renomear (ρ)

Renomear permite mudar o nome de uma relação, chamá-la por mais de um nome, ou renomear o resultado de uma operação qualquer. Sua sintaxe é:

$$\rho_{\text{nome(atributos)}}(R)$$

Considere uma relação R (A1,A2,...,An) com seus atributos. Assim, podemos, por exemplo, escrever:

$$\rho_{X}(R)$$

Renomeamos R para X, ou:

$$\rho_{X(B1,\,B2,\ldots Bn)}(R)$$

Renomear R para X, mas trocar também o nome dos seus atributos, pode ser particularmente útil para fazer a união, interseção ou diferença entre relações que tenham atributos com nomes distintos.

b) Atribuir (←)

Podemos atribuir uma relação, ou resultado de uma atribuição, para uma nova relação. Com isso, podem-se expressar consultas, de forma mais simples, minimizando a escrita

posterior que usa aquele resultado. Sua sintaxe é:

<Nova Relação> → [Relação ou operação]

Por exemplo, pode-se chamar de ListaProjetoDepto o resultado da seleção realizada sobre o produto cartesiano de Projeto e Departamento que fizemos antes:

ListaProjetoDepto → $\sigma_{\text{Projeto.codDep = Departamento.codDep}}$ (Projeto x Departamento)

Assim, seria totalmente válida a realização de qualquer outra operação sobre a nova relação ListaProjetoDepto, como se ela já existisse de forma primitiva.

4.2.6 OPERADORES DA ÁLGEBRA RELACIONAL

O conjunto completo de operadores da álgebra relacional, sua sintaxe e significado podem ser apresentados na seguinte tabela:

OPERAÇÃO	SÍMBOLO	SINTAXE
Projeção	π ("pi")	π <lista de campos> (Tabela)
Seleção/ Restrição	σ ("sigma")	σ <condição de seleção> (Tabela)
União	$\cup$	(Tabela 1) $\cup$ (Tabela 2)
Interseção	$\cap$	(Tabela 1) $\cap$ (Tabela 2)
Diferença	–	(Tabela 1) – (Tabela 2)
Produto Cartesiano	X	(Tabela 1) **X** (Tabela 2)
Junção	|X|	(Tabela 1) |**X**| <condição de junção> (Tabela 2)
Divisão	÷	(Tabela 1) ÷ (Tabela 2)
Renomeação	ρ ("rho")	ρ Nome(Tabela)
Atribuição	←	Variável ← Tabela

Figura 23 - Operadores da álgebra relacional.
Fonte: Elaborado pelo autor (Julho 2020).

4.3 INTRODUÇÃO À SQL

A Linguagem SQL é uma linguagem de trabalho com base de dados relacionais, originária dos estudos de E. F. Cood, no início dos anos 70, no século passado. Seu nome

vem das iniciais de Structures Query Language – SQL ou, em português, de Linguagem De Consulta Estruturada. Por mais que seu nome traga, explicitamente, a palavra consulta, esta linguagem tem várias outras sublinguagens embutidas, inclusive a de consulta.

Ela é utilizada para interagir com SGBD e executa várias tarefas, tais como: definir e construir a base de dados, inserir, alterar e remover registros e, após, realizar consultas em vários níveis de complexidade. Ela é derivada dos recursos da álgebra relacional e permite que todas as ações necessárias para se realizar com a base de dados possuam uma sintaxe muito parecida, facilitando o entendimento da linguagem.

A Linguagem SQL é composta por 5 (cinco) outras linguagens que realizam partes específicas do trabalho com o banco de dados. Estas sublinguagens são apresentadas na Figura 24, sendo suas funções:

a. Linguagem de Consulta de Dados – DQL: define os comandos, utilizados para realizar consultas em uma base relacional;
b. Linguagem de Manipulação de Dados – DML: define os comandos, utilizados para realizar a inserção, alteração e remoção dos registros na base;
c. Linguagem de Definição de Dados – DDL: define os comandos, utilizados para criar as tabelas e relações na base, além de alterar e remover estas tabelas.
d. Linguagem de Controle de Dados – DCL: define os comandos, utilizados para controlar o acesso ao banco de dados. São elementos desta linguagem os comandos GRANT e REVOKE que adicionam ou removem permissões de acesso à base de dados.
e. Linguagem de Transação de Dados – DTL: define os comandos, utilizados para gerenciar as transações executadas no banco de dados. Fazem parte delas, comandos com BEGIN TRANSACTION, COMMIT e ROLLBACK que controlam a execução de comandos ou grupos de comando sobre a base de dados.

Este material abordará as linguagens de definição (DDL), manipulação (DML) e consulta (DML) sobre a base de dados. As linguagens de controle (DCL) e do controle de transações (DTL) fazem parte do estudo avançado da Linguagem SQL, não sendo contempladas neste caderno.

SAIBA MAIS

Maiores informações das linguagens DTL e DCL podem ser obtidas em diversos dos livros, usados nas referências deste caderno, especialmente em materiais, como: [ELSMARI & NAVATHE, 2019], [DATE, 2004] e [CARDOSO & CARDOSO, 2013], recomendados para você aprofundar seus conhecimentos. Entretanto, faça este aprofundamento, após conhecer e dominar as outras três sublinguagens do SQL.

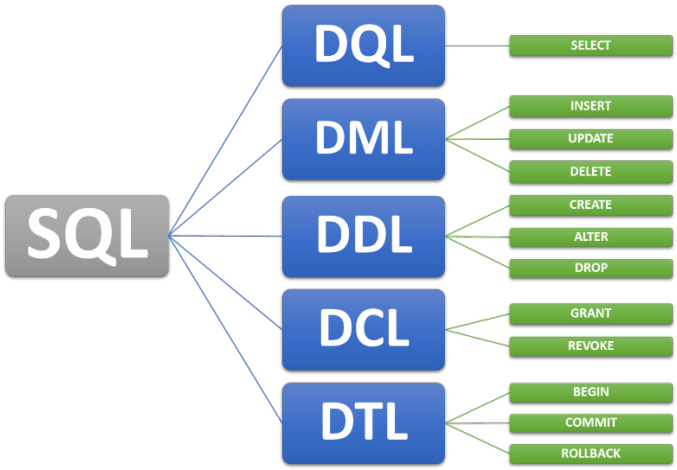


Figura 24 - Sublinguagens que compõem o SQL.
Fonte: https://www.devmedia.com.br/guia/guia-completo-de-sql/38314. Acesso em: Julho/2020.

A Linguagem SQL é uma linguagem não procedimental, na qual devemos especificar **o que** desejamos fazer e não **como** desejamos que isto seja realizado. Ela faz uma clara abstração da estrutura física dos dados e, com isso, não é necessário especificar os caminhos de acesso, ou os algoritmos que serão utilizados para realizar a tarefa.

Todas as operações realizadas com SQL são aplicadas sobre o conjunto de dados (relações e tabelas) e produzem como resposta outra relação ou tabela. Assim, o SQL procede exatamente como a álgebra relacional propõe em sua definição.

4.4 DEFINIÇÃO DE DADOS – DDL

Para realizar a definição da estrutura da base de dados existem três comandos dentro da SQL. São eles: o CREATE, o ALTER e o DROP. Vamos a cada um deles:

a) CREATE

O comando CREATE é o responsável pela criação de objetos na base de dados. Com ele, podemos criar o banco de dados, tabelas, visões, índices e outros elementos em nossa base. Sua sintaxe é:

CREATE <Elemento> nome_do_elemento

Por exemplo, podemos criar uma base de dados em nosso banco, entrando com o comando CREATE DATABASE nome_banco. Porém, de forma especial, entenderemos, neste momento, o comando CREATE TABLE. Sua sintaxe é:

```
CREATE TABLE nome_tabela (
[nome_coluna tipo_de_dado restrições],
[nome_coluna tipo_de_dado restrições],
[nome_coluna tipo_de_dado restrições],
.....
)
```

Para definirmos nossa base de dados da Figura 19, precisamos criar as relações Empregado, Projeto, Departamento e ProjetoEmpregado. Cada uma das relações, durante a fase de projeto, teve seus atributos e domínios determinados pelo projetista. Com o comando CREATE, vamos informar isto ao banco de dados. Supondo-se que o resultado do processo de projeto foram as definições:

Relação	Atributo	Tipo de Dado	Tamanho	Restrição
Empregado	Código	Inteiro		Chave primária
	Nome	Caractere	50	
	Salário	Real (decimal)	12,2	
	Idade	Inteiro		
	Cod. Departamento	Inteiro		Chave estrangeira
Projeto	Código	Inteiro		Chave primária
	Descrição	Caractere	50	
	Cod. Departamento	Inteiro		Chave estrangeira
Departamento	Código	Inteiro		Chave primária
	Descrição	Caractere	50	

Projeto_ Empregado	Cod. Projeto	Inteiro		Chave primária
	Cod. Empregado	Inteiro		Chave primária
	Data início	Data		
	Data final	Data		

Para realizar a criação destas relações com seus atributos, as implementações dos bancos de dados fornecem determinadas palavras-chave que podem variar, dependendo das circunstâncias. Este material usa as definições dos dialetos da Linguagem SQL, principalmente o que foi descrito nos padrões ANSI 2006, ANSI 2003 e ANSI 1999.

Os principais tipos de dados, presentes na Linguagem SQL, são os seguintes (existem outros tipos que você deve verificar no banco de dados que está usando):

- CHAR: tipo de caractere, com tamanho de armazenamento fixo, de acordo com a quantidade de caracteres definidos na criação da tabela;
- VARCHAR: tipo de caractere, com tamanho de armazenamento variável, de acordo com os valores inseridos pelo usuário. Neste caso, o tamanho indicado no momento da criação irá definir a quantidade máxima de caracteres a serem aceitos naquele atributo;
- INTEGER: dados do tipo inteiro (sem casas decimais). É possível indicar um tamanho em bytes para este campo e, quanto maior for o valor indicado, maior será o seu domínio.
- DECIMAL: destinado a dados do tipo numérico, com casas decimais. Neste tipo, indicamos o número total de casas do número e, destas, quantas serão usadas para armazenamento de valores decimais. Por exemplo, DECIMAL(12,2) indicar um número real, com 12 casas e, destas, duas serão destinadas para a parte decimal.
- DATE: destinados a atributos que devem armazenar datas no formato AAAA-MM-DD, ou seja, ano, mês e dia, nesta ordem.
- DATETIME: destinados a atributos com data e hora, armazenados conjuntamente.
- TIME: definem atributos do tipo tempo, contendo HH:MM:SS, que são horas, minutos e segundos.

Existem, também, restrições que podem (e devem) ser indicadas, quando da criação dos atributos. As principais são:

= > NOT NULL: indica que o atributo deverá ser, obrigatoriamente, informado pelo usuário, na hora de inserir dados na tabela. (Todos os campos, por default, aceitam valores nulos);

= > AUTO_INCREMENT: indica que o atributo será preenchido, automaticamente, com valores autoincrementados (não é possível definir um valor para este atributo na

hora de inserir dados – a cada nova inserção, o banco determinará qual o próximo valor a usar para aquele campo);

= > PRIMARY KEY: restrição de integridade que define a chave primária da tabela (se a chave for composta, os nomes devem ser separados por vírgulas);

= > UNIQUE: restrição de integridade, a qual indica que um campo não poderá receber valores repetidos na tabela (ou seja, dois registros não podem ter o mesmo valor para este campo)

= > CHECK: restrição de integridade que indica condições para o preenchimento de um campo. Por exemplo, podemos indicar, para um campo do tipo idade, que somente aceitaremos idades maiores ou iguais a 18 anos, inserindo CHECK(idade >= 18);

= > In (conjunto de valores) – indica que o valor de um determinado atributo deve estar presente no conjunto de valores definido, para que os dados possam ser inseridos na tabela;

Caso alguma das restrições acima (ou outras, presentes no seu banco de dados) seja infringida no momento da inserção ou alteração de dados, a operação que estiver sendo realizada falhará e retornará na forma de erro.

Além desses elementos, também existem restrições explícitas, como a indicação de chaves estrangeiras, e o que deve ser realizado, em caso de ações sobre elas, como a remoção. Para isso, usamos o comando CONSTRAINT, o qual tem por sintaxe:

```
CONSTRAINT nome_restrição
FOREIGN KEY (atributo_1, atributo_2, ... atributo_n)
REFERENCES Relação_Pai (atributo_1, atributo_2, ... atributo_n)
[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

Com o comando CONSTRAINT, devemos indicar qual é o atributo, que será a chave estrangeira, à qual tabela pai ela se refere e a qual atributo, sendo esta a sua forma base. Podemos, também, indicar o que deverá acontecer em caso de remoção (ON DELETE) ou alteração (ON UPDATE). Para isso, indicamos que nada deve acontecer (NO ACTION), o comando deve acontecer em cascata (CASCADE), ser marcado como nulo (SET NULL), ou ser marcado com o valor padrão (SET DEFAULT) naquele campo.

Usando as definições de projeto, apresentadas anteriormente, podemos criar nossa base de dados com os seguintes comandos:

```
CREATE TABLE Departamento(
        codigo INTEGER NOT NULL AUTO_INCREMENT,
        descricao VARCHAR(50),
        PRIMARY KEY( codigo )
);

CREATE TABLE Empregado(
        codigo INTEGER NOT NULL AUTO_INCREMENT,
        nome VARCHAR (50),
        salario DECIMAL(12,2),
        idade INTEGER,
        codDep INTEGER NOT NULL,
        PRIMARY KEY( codigo ),
        CONSTRAINT fk_depto FOREIGN KEY (codDep) REFERENCES Departamento
(codigo)
);

CREATE TABLE Projeto(
        codigo INTEGER NOT NULL AUTO_INCREMENT,
        descricao VARCHAR (50),
        codDep INTEGER NOT NULL,
        PRIMARY KEY( codigo ),
        CONSTRAINT fk_depto FOREIGN KEY (codDep) REFERENCES Departamento
(codigo)
);

CREATE TABLE ProjetoEmpregado(
        codProj INTEGER NOT NULL,
        codEmp INTEGER NOT NULL,
        dataIn DATE,
        dataFin DATE,
        PRIMARY KEY( codProj, codEmp )
        CONSTRAINT fk_proj FOREIGN KEY (codProj) REFERENCES Projeto (codigo)
        CONSTRAINT fk_demp FOREIGN KEY (codEmp) REFERENCES Empregado
(codigo)
);
```

As restrições aplicadas para chaves estrangeiras em nossas relações impõem uma ordem para a criação das tabelas. Por exemplo, se desejarmos criar a tabela Empregado, antes da tabela Departamento, receberemos um erro na criação, pois há, em Empregado, uma referência para Departamento. Outro detalhe é que não foi indicada nenhuma ação especial a ser tomada, em caso de violação da restrição da chave estrangeira. Isso acontece, quando a tabela Empregado, por exemplo, aponta para um valor de departamento que não existe. Como omitimos a ação, caso este atributo seja inválido, a operação que pode causar, será negada.

Outra questão relevante é quanto ao campo para datas (DATE ou DATETIME). É importante observar que ele é armazenado, sempre com o formato Ano-Mês-Dia, nesta ordem, o que é necessário, para permitir que possamos ordenar um atributo deste tipo por data[9].

b) ALTER TABLE

O comando ALTER é utilizado para alterar alguma definição que foi atribuída a um objeto, anteriormente. Por exemplo, podemos mudar o nome ou o tipo de dado de um atributo, ou dar a ele uma restrição qualquer. Para isso, o comando ALTER vem associado ao tipo de alteração que deverá ser realizada (ADD, DROP ou CHANGE), mas sempre respeitando qualquer restrição, a qual tenha sido atribuída anteriormente.

Em especial, vamos detalhar o comando ALTER TABLE que possui as seguintes variações:

```
ALTER TABLE nome_tabela ADD novas_colunas | novas_restrições

ALTER TABLE nome_tabela CHANGE|MODIFY definição_atributo

ALTER TABLE nome_tabela DROP atributo
```

Com o uso do ADD, podemos inserir novos atributos na tabela, ou dar novas restrições a um atributo existente. Podemos, por exemplo, incluir um atributo data_nascimento, na tabela Empregado, entrando com o comando:

9 Observe que nosso formato de data DD-MM-AAAA não é ordenável, sem separarmos cada um dos atributos. Nele 10-12-2020 será menor que 15-01-2020, a não ser que separemos e analisemos cada uma das partes isoladamente. Isso não acontece com o formato AAAA-MM-DD.

ALTER TABLE Empregado ADD data_nascimento DATE NOT NULL

Usando o CHAGE (ou MODIFY em alguns bancos), é possível alterar a definição de algum atributo existente. Por exemplo, vamos supor que precisemos aumentar o número de caracteres do atributo descrição da tabela Projeto, trocando seu nome para nome_projeto. Para isso, devemos entrar com o comando:

ALTER TABLE Projeto CHANGE descricao nome_projeto VARCHAR(100)

Finalmente, podemos eliminar um atributo de uma tabela. Por exemplo, vamos remover o atributo idade, da tabela Empregado:

ALTER TABLE Empregado DROP idade

c) DROP TABLE

O comando DROP já foi parcialmente introduzido na descrição anterior, com o ALTER. Por outro lado, seu uso pleno, como indicado aqui, serve para eliminarmos toda uma relação ou tabela da base de dados.

Por exemplo, caso seja necessário remover a relação Departamento, da nossa base de dados, podemos entrar com o comando;

DROP TABLE Departamento

4.5 MANIPULAÇÃO DE DADOS - DML

A linguagem de manipulação de dados, dentro da SQL, é usada para manipular (inserir, alterar ou remover) tuplas de uma relação. Ela é constituída por três comandos: INSERT, UPDATE e DELETE.

a) INSERT

O comando INSERT é usado em SQL para inserirmos dados dentro de uma relação qualquer. Sua sintaxe é:

INSERT INTO nome_tabela [(atributo1, atributo2, ...)]
VALUES (valor1, valor2, ...)

Com o uso de comando INSERT, podemos povoar nossas tabelas da base de dados, no exemplo que estamos usando. Para inserir os dados da tabela Departamento, devemos fazer:

INSERT INTO Departamento VALUES ('1', 'Pesquisa);
INSERT INTO Departamento VALUES ('2', 'Desenvolvimento');

Observe que não utilizamos, nos exemplos acima, a parte opcional do comando INSERT que descreve quais colunas deverão ser utilizadas, para cada um dos valores expressos na sequência. Isso acontece, porque usam-se todos dos atributos da relação, e na mesma ordem. Quando mudamos a ordem dos atributos na entrada, ou não fornecemos valores para todos eles, devemos descrever quais são os atributos que estamos usando. Por exemplo:

INSERT INTO Projeto (codigo, codDep, descricao) VALUES ('', '1', 'AATOM');

Mais um detalhe importante, como declaramos o atributo código como AUTO_INCREMENT, ao passarmos um valor vazio ou nulo a este atributo, o banco de dados buscará o próximo valor válido disponível e atribuirá a este campo. Como é a nossa primeira inserção na tabela, o atributo receberá o valor 1. Outro detalhe é que trocamos a ordem dos campos, veja que fornecemos o código do departamento antes da descrição, e isso foi indicado pela ordem das colunas que foi colocada no comando.

b) UPDATE

Podemos alterar o valor de um ou mais atributos de uma relação, usando o comando UPDATE. Sua sintaxe é dada por:

UPDATE nome_tabela SET atributo = valor, ...
[WHERE condição]

O comando UPDATE é condicional, podendo ser utilizado ou não. Caso uma condição seja expressa no comando, a alteração indicada será realizada apenas nas tuplas que atenderem à condição estabelecida. Caso nenhuma condição seja indicada, todas as tuplas da relação serão afetadas pela alteração.

Por exemplo, podemos alterar a idade do empregado Maria, para 28 anos, fazendo:

UPDATE Empregado SET idade=28 WHERE codigo=202

No comando UPDATE, podemos indicar operações matemáticas, a serem utilizadas na atribuição. Da mesma forma que podemos usar qualquer operador relacional (<, <=, >, >=, <>, =), ou lógico (AND, OR e NOT). Caso aumentemos em 10% o salário de todos os empregados que ganham menos de R$ 2.000,00, ou tenham mais de 40 anos, com o comando:

```
UPDATE Empregado SET salario = salario * 1.10 WHERE
salario < 2000 OR idade > 40
```

Após esse comando, todos os empregados dos nossos exemplos terão seu salário aumentado, exceto a Maria, que ganha mais de 2000 e tem menos de 40 anos, ou seja, não atende a nenhuma das condições exigidas para o aumento.

c) DELETE

O comando DELETE completa a lista de comandos da DML, removendo uma ou mais tuplas da relação indicada. Sua sintaxe é:

DELETE FROM nome_tabela [WHERE condição]

Após o uso do DELETE, todas as linhas que respeitarem a condição imposta, inclusive com operadores relacionais e lógicos inseridos (como no UPDATE), serão removidas da relação. Caso nenhuma condição seja imposta, **todas** as tuplas da tabela serão removidas (Cuidado!!!!).

Por exemplo, vamos remover todos os empregados que estejam alocados ao departamento de código 2:

```
DELETE FROM Empregado WHERE codDep = 2
```

4.6 CONSULTA DE DADOS - DQL

A ação mais comum, em uma base de dados, é a consulta às informações presentes nesta base. Embora seja a ação mais frequente, ela é formada, dentro da linguagem SQL, por um único comando: SELECT.

Fazendo uma relação entre o SQL e a álgebra relacional, o comando SELECT é usado

para fazer seleção, projeção, produto cartesiano, junções e várias outras operações presentes na álgebra.

A sintaxe deste comando é:

SELECT [DISTINCT] atributo1, atributo2, ... | * FROM tabela1, tabela2, ...
[WHERE condição]
[GROUP BY atributo1, atributo2,...]
[ORDER BY atributo1 ASC|DESC, atributo2 ASC|DESC,....]
[HAVING condição]
[LIMIT início, final]

Como o SELECT permite combinarmos várias tabelas, atributos e condições, ele apresenta uma grande variedade de combinações que produzem resultados bem específicos, dos mais simples aos mais complicados.

a) Seleção Simples

O formato mais simples de um comando SELECT é quando o usamos, para fazer seleções ou projeções em uma tabela única. Para isso, utiliza-se a lista de atributos que desejamos selecionar (para a projeção) ou um * (asterisco), para indicar que desejamos todos os atributos e expressamos, na cláusula WHERE, qual a condição de seleção. Caso o WHERE não esteja presente, todas as tuplas da relação serão retornadas.

Vamos a alguns exemplos de seleções simples:

Descrição	Comando
Todos os empregados com salário maior que 2.500,00.	SELECT * FROM Empregado WHERE salario > 2500
Apenas o nome dos empregados, com idade entre 20 e 40 anos.	SELECT nome FROM Empregado WHERE idade >= 20 AND idade <= 40
Código e descrição dos projetos do departamento 2.	SELECT codigo, descricao FROM Projeto WHERE codDep = 2

b) Expressões e Constantes

Na lista de atributos de um SELECT, podemos colocar expressões aritméticas entre os atributos da relação e, até mesmo, mostrar alguma constante. Podemos, por exemplo, escrever um SELECT como:

```
SELECT nome, “Salário”, salario*1.5 FROM Empregado
```

Este comando mostrará o nome de todos os Empregados e, ao lado de cada nome, virá a palavra (constante) “Salário”, seguida do valor do salário, aumentado em 50%. Operações matemáticas tradicionais, como soma, subtração, multiplicação e divisão podem ser utilizadas com os atributos. Existem, também, outros elementos que serão descritos na secção 4.7, logo a seguir.

c) Ordenação

Após os comandos de um SELECT, podemos definir o critério de ordenação, o qual desejamos usar no resultado, devendo-se incluir, ao final do comando, a cláusula ORDER BY. Nesta cláusula, indicamos os atributos a serem usados na ordenação, bem como se o desejamos em ordem crescente (padrão, se não indicarmos), ou decrescente.

Por exemplo, vamos listar todos os empregados cadastrados ordenados por salário, de forma crescente e, caso haja empate em dois salários, devemos usar como critério a idade de forma decrescente:

```
SELECT * FROM Empregado ORDER BY salario ASC, idade DESC
```

Como mencionado acima, o critério ASC é opcional e será usado, em caso de nada ser indicado no SELECT.

d) Seleções compostas

Nas seleções compostas, indicamos mais de uma tabela na lista FROM de um SELECT. É muito importante salientar que, ao indicarmos duas (ou mais) relações no SELECT, será realizado o produto cartesiano (da álgebra relacional) com estas duas tabelas. Caso queiramos fazer uma junção (também no conceito da álgebra) devemos indicar o filtro da seleção dos elementos desejados.

Por exemplo, caso queiramos listar todos os Empregados, com o nome do departamento onde eles estão alocados, ordenados por nome, podemos entrar com o comando:

```
SELECT * FROM Empregado,Departamento
WHERE Empregado.codDep = Departamento.codigo
ORDER BY Empregado.nome
```

Observe que indicamos o nome de cada uma das relações, antes do nome do atributo. Isso é necessário, sempre que tenhamos atributos com o mesmo nome, nas diferentes

relações usadas. Caso os nomes sejam todos distintos entre as relações, não precisamos indicar o predicado de distinção.

Se não desejarmos escrever o nome completo da relação, podemos usar o conceito de aliases, presente na SQL. Aliases são "apelidos" dados às relações ou aos atributos, em uma seleção. Por exemplo, na seleção acima, podemos indicar que a relação Empregado será conhecida como E, e Departamento, como D:

```
SELECT E.nome, E.salario, D.descricao
FROM Empregado AS E, Departamento AS D
WHERE E.codDep = D.codigo  ORDER BY E.nome
```

4.7 FUNÇÕES E PREDICADOS

Ao utilizarmos o comando SELECT, podemos usar algumas funções que são separadas por natureza da sua ação. Vejamos algumas das principais:

a) Funções para valores simples

- **ABS(n)**: Devolve o valor absoluto de (n).
- **CEIL(n)**: Obtém o valor inteiro, imediatamente superior ou igual a "n".
- **FLOOT(n)**: Devolve o valor inteiro, imediatamente inferior ou igual a "n".
- **MOD (m, n)**: Devolve o resto, resultante de dividir "m" entre "n".
- **NVL (valor, expressão)**: Substitui um valor nulo por outro valor.
- **POWER (m, exponente)**: Calcula a potência de um número.
- **ROUND (numero [, m])**: Arredonda números, com o número de dígitos de precisão indicados.
- **SQRT(n)**: Devolve a raiz quadrada de "n".
- **TRUNC (numero, [m])**: Trunca números, para que tenham uma certa quantidade de dígitos de precisão.

b) Funções para valores agregados

- **AVG(n)**: Calcula o valor médio de "n", ignorando os valores nulos.
- **COUNT (* | Expressão)**: Conta o número de vezes que a expressão avalia algum dado com valor não nulo. A opção "*" conta todas as filas selecionadas.
- **MAX (expressão)**: Calcula o máximo.

- **MIN (expressão)**: Calcula o mínimo.
- **SUM (expressão)**: Obtém a soma dos valores da expressão.
- **GREATEST (valor1, valor2...)**: Obtém o maior valor da lista.
- **LEAST (valor1, valor2...)**: Obtém o menor valor da lista.

c) Funções para caracteres

- **CONCAT (cad1, cad2)**: Devolve "cad1" concatenada com "cad2".
- **LOWER (cad)**: Devolve a cadeia "cad" em minúsculas.
- **UPPER (cad)**: Devolve a cadeia "cad" em maiúsculas.
- **LPAD (cad1, n)**: Adiciona caracteres à esquerda da cadeia, até que tenha um certo tamanho n.
- **RPAD (cad1, n)**: Adiciona caracteres à direita, até que tenha um certo tamanho n.
- **LTRIM (cad [,set])**: Suprime um conjunto de caracteres à esquerda da cadeia.
- **RTRIM (cad [,set])**: Suprime um conjunto de caracteres à direita da cadeia.
- **REPLACE (cad, cadeia_busca [, cadeia_substitucao])**: Substitui um caractere, ou caracteres, de uma cadeia com 0 ou mais caracteres.
- **SUBSTR (cad, m [,n])**: Obtém parte de uma cadeia.
- **LENGTH (cad)**: Devolve o número de caracteres de cad.

d) Funções para datas

- **SYSDATE**: Devolve a data do sistema.
- **ADD_MONTHS (data, n)**: Devolve a data "data", incrementada em "n" meses.
- **LASTDAY (data)**: Devolve a data do último dia do mês que contém "data".
- **MONTHS_BETWEEN (data1, data2)**: Devolve a diferença, em meses, entre as datas "data1" e "data2".
- **NEXT_DAY (data, cad)**: Devolve a data do primeiro dia da semana, indicado por "cad", depois da data indicada por "data".
- **YEAR(data)/MONTH(data)/DAY(data)**: Devolve o ano, mês ou dia da data indicada.

Vamos a alguns exemplos, usando essas funções:

SELECT count(*) FROM Empregado WHERE codDep=1

a. Contará o número de empregados do departamento 1.

SELECT max(salario) FROM Empregado

b. Apresentará o maior salário presente na tabela Empregado.

SELECT agv(salario) FROM Empregado WHERE codDep = 2

c. Mostrará a média dos salários dos empregados do departamento 2.

SELECTE.nome,months_between(dataIn,dataFin)asTempoFROMProjetoEmpregado as P, Empregado as E WHERE P.codEmp = E.codigo AND dataFim<>null

d. Mostrará o nome e o tempo que cada um dos empregados destinou aos projetos onde trabalhou. Observe que foi usado um alias (tempo), para o resultado da função.

Outro ponto a chamar a atenção é quanto ao uso de letras maiúsculas e minúsculas nos comandos e funções do SQL. A linguagem SQL não é sensível à letra, e podemos usar maiúsculas e minúsculas (ou uma mescla delas) na expressão de qualquer comando.

Além das funções, existem predicados que podemos usar no SELECT. Normalmente, os predicados são usados na cláusula WHERE. Os predicados mais comuns são os seguintes:

1- LIKE

A cláusula LIKE é utilizada para fazer busca de caracteres por semelhança ou proximidade. Com ela podemos, por exemplo, procurar todos os nomes que iniciam por um dado caractere ou terminam por outros.

Para uso do LIKE podemos usar os símbolos % (porcentagem) e _ (underscore) para indicar o que desejamos. A porcentagem (%) indica que aceitamos uma sequência de caracteres, enquanto o underscore (_) é usado para indicar um único caractere. Observe as seleções:

SELECT * FROM Empregado WHERE nome LIKE “MA%”

e. Lista todos os empregados, cujo nome inicie por MA, não importando o que venha depois (%)

SELECT * FROM Projeto WHERE descricao LIKE “_ES%”

f. Lista todos os projetos, cujo nome comece por qualquer caractere (apenas um), tenha ES depois, obrigatoriamente, seguidos por qualquer outro texto.

Existem outras combinações que podemos fazer, usando [] (colchetes) e ^ (acento circunflexo). Os colchetes indicam uma combinação possível de caracteres, já o circunflexo indica a negação. Por exemplo:

- Como encontrar todos os nomes que começam com A ou B?
  Usar **LIKE '[AB]%'**
- Como encontrar todos os nomes que começam com as letras de A até E?
  Usar **LIKE '[A-E]%'**
- E todos os nomes que não iniciam com V?
  Usar **LIKE '[^V]%'**

2- BETWEEN

A cláusula BETWEEN é usada para buscar elementos dentro de um intervalo de valores. Para isso, indicamos os limites desejados dentro do SELECT. Por exemplo, para listarmos todos os empregados com idades entre 20 e 40 anos, podemos escrever:

```
SELECT * FROM Empregado WHERE idade BETWEEN 20 AND 40
```

3- NULL

A palavra NULL pode ser utilizada como um seletor válido dentro das condições do SELECT. Podemos, inclusive, combiná-la com a negação NOT. Assim, podemos selecionar:

```
SELECT * FROM ProjetoEmpregado WHERE dataFin IS NULL
```

Ou

```
SELECT * FROM ProjetoEmpregado WHERE dataFin IS NOT NULL
```

Essas seleções mostrarão os elementos da relação que tenham (IS) ou não (NOT IS), a data final igual a nulo, ou seja, que não tenha sido preenchida.

4- IN

A cláusula IN permite selecionar elementos que pertençam ou não a uma lista de valores indicados dentro dela. Por exemplo, para listarmos todos os empregados que tenham as idades de 25, 28 e 32 anos, podemos escrever:

```
SELECT * FROM Empregado WHERE idade IN (25, 28, 32)
```

A mesma situação seria possível com o uso do NOT, junto com o IN, para indicar uma seleção de elementos que não esteja dentro da lista dada. Uma outra possibilidade, muito

útil, é o uso do IN (com ou sem o NOT) para realizar uma *subquery*, que é uma segunda consulta, dentro da principal.

Por exemplo, vamos listar todos os empregados que iniciaram em algum projeto durante o ano de 2019:

```
SELECT * FROM Empregado WHERE codigo IN (SELECT codEmp FROM
ProjetoEmpregado WHERE YEAR(dataIn)=2019)
```

5- ALL e ANY

Os predicados ALL e ANY são usados com subconsultas ou *subqueries* (como do IN acima) e retornam verdade ou falso. O ALL retorna verdade, se todos os valores da subconsulta confirmam a condição, enquanto o ANY retorna verdade de algum valor que fizer isto.

Por exemplo, vamos listar os empregados, cujos salários sejam superiores a todos os salários dos empregados do departamento 1:

```
SELECT nome, descricao FROM  Empregado, Departamento
WHERE Empregado.codDep = Departamento.codigo
AND salario > ALL (SELECT salario FROM Empregado WHERE codDep = 1)
```

No lugar do ALL, podemos usar o ANY, para buscarmos a situação, em que o salário buscado seja maior do que qualquer um dos salários do departamento 1.

6- EXISTS e NOT EXISTS

EXISTS é uma cláusula de uso semelhante ao ANY e ALL acima. Neste caso, a condição será verdadeira, caso a subconsulta não seja vazia. Por exemplo, podemos listar todos os departamentos que tenham, pelo menos, um funcionário alocado:

```
SELECT descricao FROM Departamento as D
WHERE EXISTS
(SELECT  * FROM Empregado as E WHERE D.codigo = E.codDep)
```

7- DISTINCT

O uso do SELECT pode resultar em tuplas duplicadas em uma seleção ou projeção. A cláusula DISTINCT é utilizada para apresentar apenas elementos diferentes (distintos)

no resultado do SELECT, isto é, DISTINCT elimina tuplas duplicadas do resultado. Para usá-la devemos declarar:

SELECT DISTINCT atributo1, atributo2, ... FROM nome_tabela

Por exemplo, para listarmos apenas os empregados que tenham nomes diferentes, podemos escrever:

SELECT DISTINCT nome FROM Empregado

8- GROUP BY

O uso da cláusula GROUP BY permite agrupar tuplas, de acordo com algum critério previamente definido. Por exemplo, para listarmos a quantidade de empregados de cada um dos departamentos, podemos escrever:

SELECT count(*) FROM Empregado GROUP BY codDep

9- HAVING

A cláusula HAVING pode ser utilizada apenas com o GROUP BY. Ela estabelece uma condição para fazer o agrupamento. Por exemplo, no SELECT acima, podemos desejar listar apenas aquelas situações com departamentos que tenham mais do que 5 empregados. Para isso, devemos escrever:

SELECT count(*) as total FROM Empregado
GROUP BY codDep HAVING total > 5

4.8 UNIÃO, INTERSECÇÃO E DIFERENÇA

Para realizarmos as operações de conjuntos em SQL, devemos ter relações com os mesmos atributos e domínios, assim como na álgebra relacional. Se as relações não forem assim, podemos usar as projeções e/ou funções de conversão para termos relações com os mesmos elementos nas operações.

a) UNIÃO

A operação de união cria uma nova relação, contendo a junção das tuplas de duas seleções diferentes, mas sem elementos repetidos. Por exemplo, vamos supor uma base de dados, contendo os cadastros de clientes e fornecedores, com nome e endereço deles, mas com elementos que estão nos dois cadastros (são duplicados). Para listar os nomes e endereços deles, para fazermos uma mala direta, podemos escrever:

```
SELECT nome, endereço FROM Cliente
UNION
SELECT nome, endereço FROM Fornecedor
```

b) INTERSECÇÃO

A operação de intersecção pode ser expressa, em alguns bancos de dados, por uma cláusula nova, chamada INTERSECT. Entretanto, para fazer isso, devemos usar o predicado IN, descrito acima, com uma subconsulta. Por exemplo, para listarmos o nome e o endereço, apenas dos clientes que são também fornecedores, devemos escrever:

```
SELECT nome, endereço FROM Cliente
IN
( SELECT nome, endereço FROM Fornecedor )
```

c) DIFERENÇA

A mesma lógica da intersecção vale para a diferença. Para fazermos a diferença entre os resultados de duas relações, usamos o NOT IN. Podemos usar, caso o banco também aceite, a cláusula EXCEPT, mas ela não estará presente em muitas implementações. Por exemplo, para listarmos apenas os clientes que não sejam fornecedores, devemos escrever:

```
SELECT nome, endereço FROM Cliente
NOT IN
( SELECT nome, endereço FROM Fornecedor )
```

4.9 JUNÇÕES

As junções entre duas tabelas, na Linguagem SQL, podem ser realizadas, usando o produto cartesiano entre elas e, posteriormente, filtrando os resultados. Além disso, a SQL fornece diversos tipos de junções ou *joins*, como são conhecidos, que atendem às necessidades de combinações entre relações.

Conforme podemos observar na Figura 25, existem sete tipos diferentes de junções. Cada uma delas usa duas relações A e B, e a área em vermelho, presente na figura, mostra onde estão localizados elementos do resultado.

O uso do produto cartesiano e da seleção, além de ineficiente, apresenta apenas aqueles resultados que tenham o filtro aceito. Por exemplo, se juntarmos os Empregados e os Departamentos, em nosso exemplo, esta situação iria mostrar apenas os empregados que estiverem alocados em algum departamento. Mas se tivermos empregados sem departamento, ou departamentos sem empregados, como mostrá-los? Para isso é que existem as junções.

Em primeiro lugar, temos as junções à esquerda e à direita, os *left joins* e *right joins*. Nestas junções, o resultado permite apresentar, inclusive, elementos que não tenham a condição aceita.

Por exemplo, se fizermos uma junção à esquerda entre Empregado e Departamento, listaremos os empregados com os seus departamentos, mas também serão apresentados empregados que não tenham departamento para eles. Se fizermos uma junção à direita, listaremos os departamentos todos, inclusive aqueles que não tenham empregados.

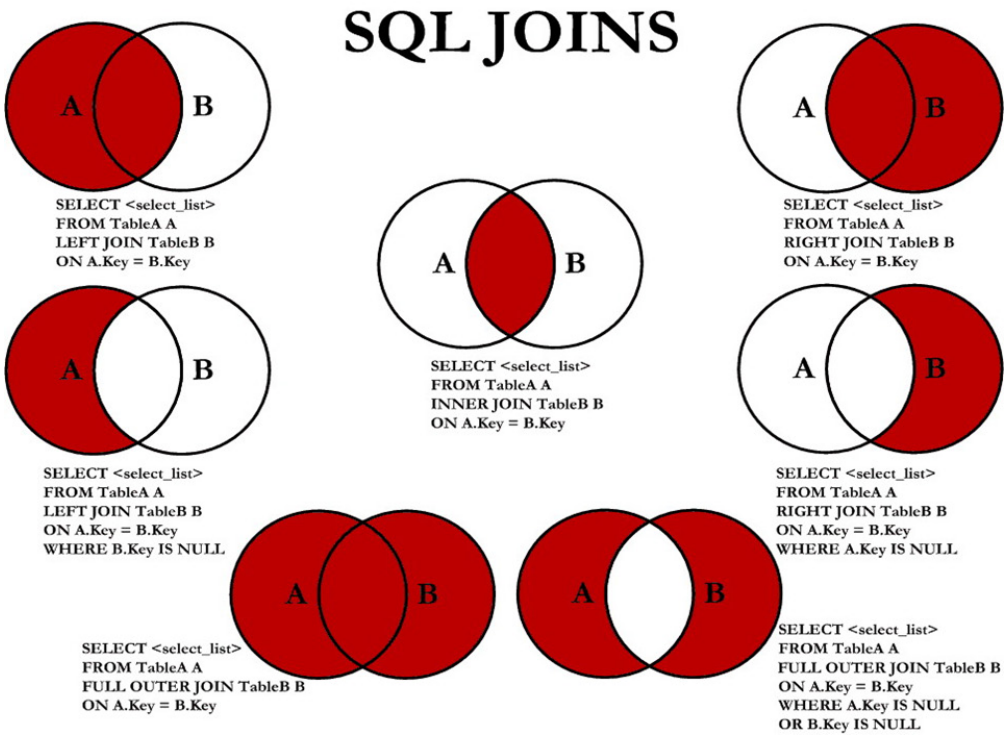


Figura 25 - Junções entre relações usando SQL.
Fonte: https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins. Acesso em: Julho/2020.

Existem, também, as junções por dentro (*inner join*) e por fora (*outer join*) que privilegiam a parte da intersecção entre os conjuntos (como na junção por produto cartesiano) ou pela parte de fora das relações (parte exclusiva de cada uma delas).

4.10 VISÕES

As visões ou *views* na SQL constituem um importante recurso de segurança ou simplificação em nossas consultas. A visão é uma tabela virtual, que não existe na prática, mas permite que façamos uso dela como se fosse uma relação normal para SELECT's posteriores.

As visões são ajustadas e complementadas em tempo de execução pelo banco de dados, sem que seja necessária qualquer ação por parte dos usuários. A visão pode ser interpretada como sendo o resultado de uma seleção qualquer que foi previamente armazenada para uso futuro.

As vantagens de uso de *views* são as seguintes:

- **Reuso**: as *views* são objetos de caráter permanente. Pensando pelo lado produtivo,

é excelente, já que elas podem ser lidas por vários usuários, simultaneamente.

- **Segurança**: as *views* permitem que ocultemos determinadas colunas de uma tabela. Para isso, basta criarmos uma *view*, com as colunas que acharmos necessárias, as quais sejam exibidas e disponibilizadas ao usuário.
- **Simplificação do código**: as *views* nos permitem criar um código de programação muito mais limpo, na medida em que podem conter um SELECT complexo. Assim, **criar *views*** para os programadores, a fim de poupá-los do trabalho de criar SELECT's, é uma forma de aumentar a produtividade da equipe de desenvolvimento.

A sintaxe de uma visão é a seguinte:

CREATE VIEW nome_visão AS [COMANDO DE SELEÇÃO]

Vamos considerar uma situação, em nosso exemplo, na qual existam usuários que necessitam ter acesso aos dados dos Empregados, mas não podem, por segurança ou por não haver necessidade, visualizar os salários dos demais empregados. Então, podemos criar uma visão destinada a estes usuários:

```
CREATE VIEW Empregado2 AS
SELECT codigo, nome, idade, descricao FROM Empregado,Departamento
WHERE Empregado.codDep = Departamento.codigo
```

Uma vez criada a visão, podemos fazer seleções, como:

```
SELECT * FROM Empregado2 WHERE idade > 30
```

Caso, em algum momento futuro, um novo empregado seja inserido na relação Empregado, a visão Empregado2 será adaptada com os novos dados, automaticamente, sem que a visão tenha que ser atualizada.

Por outro lado, uma visão não permite alteração nos seus dados. Desta forma, não é possível fazer INSERT, UPDATE ou DELETE dentro de uma visão, apenas nas tabelas originais.

Em outra situação exemplo, podemos definir uma *view*, contendo SELECT com várias tabelas. Podemos definir uma visão, incluindo apenas os empregados que ganhem mais de R$ 10.000,00, com o comando:

```
CREATE VIEW ProjetoAltoEscalao AS
SELECT E.codigo, E.none, E.salario, P.descricao
FROM Empregado E, Projeto P, ProjetoEmpregado PE
WHERE E.salario > 10000 AND E.codigo=PE.codEmp AND PE.codProj=P.codigo
```

Quando não desejamos mais, ou não precisamos da visão, ela pode ser removida por um comando DROP:

DROP VIEW nome_visao

4.11 ÍNDICES

Ao criarmos uma tabela ou relação em uma base de dados relacional, o SGBD criará diversos índices, cuja finalidade é agilizar a consulta aos dados armazenados. Um índice, originado automaticamente por SQL, é para o atributo que for definido como chave primária de uma relação.

Ao realizarmos a inserção, alteração ou deleção de dados de uma relação, o SGBD ajusta todos os índices, os quais foram criados para esta relação, o que gasta um bom tempo de processamento, mas é relativizado, pois um SGBD realiza muito mais consultas do que manipulação de dados. Desta forma, o tempo gasto na organização dos índices é ganho de volta, com consultas realizadas de forma mais rápida.

No entanto, existem situações, nas quais os índices criados pelo SGBD não são suficientes para garantir um bom desempenho das consultas na base de dados. Quando isso acontece, podemos criar novos índices que auxiliam nas consultas. A sintaxe deste comando é:

CREATE INDEX nome_indice ON nome_tabela (atributo1, atributo2, ...)

Vamos considerar, em nosso exemplo, que precisamos realizar muitas consultas, e com muita frequência, sobre os nomes dos Empregados. Como isso está impactando no desempenho da nossa aplicação, podemos criar um índice para auxiliar nas consultas:

```
CREATE INDEX Nome_Emp ON Empregado(nome)
```

SAIBA MAIS

Busque maiores informações sobre a Linguagem SQL e seu uso, nos vídeos: https://www.youtube.com/watch?v=sPqDQN-WhjQ (DevMedia), https://www.youtube.com/watch?v=2ahAwcugixI (Juracy de Almeida) e https://www.youtube.com/watch?-v=hTCtvfw0Vx0 (Harlley Oliveira).

Estes não são os únicos, mas servem de ponto de partida para aprofundamentos. Outra dica interessante é acessar vídeos do uso de SQL, dentro do banco de dados que foi adotado na sua aplicação: Mysql, SQL Server, Postgres, Oracle e vários outros. Para cada um deles você encontrará muita informação específica na internet. O mesmo vale para a linguagem de programação que você escolheu para sua aplicação. Busque como acessar e usar bases de dados em Java, Php ou outra linguagem qualquer na rede.

CONSIDERAÇÕES FINAIS

A Linguagem SQL é a forma de acesso mais comum e importante, quando falamos de bancos de dados relacionais, como por exemplo, MySQL (MariaDB), Microsoft SQL Server, Oracle, Postgres e inúmeros outros. Em todos eles, ela é a principal linguagem (e única, em muitos casos) para manipular as informações armazenadas.

Nesta unidade, conhecemos os recursos básicos desta linguagem, além de como utilizá-los para criar as nossas relações (CREATE), removê-las (DROP), manipular dados (INSERT, UPDATE e DELETE) e consultar (SELECT) as informações. Descobrimos, também, alguns outros recursos da SQL, como junções, visões e consultas avançadas nos dados.

A Linguagem SQL é muito ampla, e vários dos seus recursos não foram abordados aqui, como por exemplo, controle de permissões de usuários e controle de transações. Caso você tenha interesse nestes assuntos, nas referências da unidade, encontrará muita informação relevante sobre os temas, e mesmo sobre aqueles que tratamos nesta unidade.

EXERCÍCIO FINAL

1. Com base nas operações relacionais, e considerando relações R, R1 e/ou R2, analise as operações listadas na coluna da esquerda e relacione-as às suas propostas, apresentadas na coluna da direita. Em seguida, marque a resposta que reflete esta associação.

I. Seleção II. Projeção III. Junção IV. Diferença V. Produto Cartesiano	( ) Produz todas as combinações de tuplas de R1 e R2 que satisfizerem uma condição de junção apenas com as comparações de igualdade. ( ) Produz uma relação que tem os atributos de R1 e R2 e inclui, como tuplas, todas as possíveis combinações de tuplas de R1 e R2. ( ) Produz uma relação que inclui todas as tuplas em R1, que não estão em R2; R1 e R2 devem apresentar uma união compatível. ( ) Produz todas as tuplas de R que satisfizerem à condição especificada. ( ) Produz uma nova relação, com apenas alguns dos atributos de R. e remove as tuplas repetidas.

a) III, IV, V, I, II.

b) III, V, IV, I, II.

c) V, III, I, IV, II.

d) V, III, IV, II, I.

e) III, V, IV, II, I.

2. Considere a seguinte tabela de um banco de dados relacional:

Cliente (CPF, Nome, Fone, EnD).

O comando SQL, para obter o Nome dos clientes, cujo campo Fone tenha o valor "nulo", é:

a) SELECT Nome, Fone (NULL) FROM Cliente

b) SELECT Nome FROM Cliente WHERE Fone = "NULL"

c) SELECT Nome (Fone NULL) FROM Cliente

d) SELECT Nome FROM Cliente WHERE Fone IS NULL

e) SELECT Nome FROM Cliente WHERE Fone LIKE "NULL"

3. Considere as seguintes afirmações, a respeito da linguagem SQL.

I. Na cláusula *order by*, para especificar a forma de ordenação, devemos indicar *Desc* para ordem descendente e *Asc* para ordem ascendente.

II. A operação de conjuntos *union*, automaticamente elimina as repetições, ao contrário da operação *select*.

III. No uso de SQL, embutida em programas escritos em outras linguagens (chamadas linguagens hospedeiras), todo o processamento da consulta é feito pelo banco de dados, e o resultado da consulta fica disponível, para que o programa possa processar uma relação por vez, exatamente como acontece com SQL interativa.

Quais estão corretas?

a) I, II e III.

b) Apenas II.

c) Apenas I e II.

d) Apenas I e III.

e) Apenas II e III.

4. Considere o seguinte esquema de dados relacional:

Cargo (CodCargo, NomeCargo)
Empregado (MatEmp, NomeEmp, Salario,CodCargo)

Considere a consulta abaixo, escrita em SQL, criada a partir do esquema relacional anterior:

```
SELECT C.CodCargo, C.NomeCargo, SUM(E.Salario)
FROM Cargo C, Empregado E
WHERE C.CodCargo=E.CodCargo
GROUP BY C.CodCargo, C.NomeCargo
HAVING COUNT(*)>2 AND AVG(E.Salario)>100
```

A consulta acima obtém o seguinte resultado:

a) Para cada cargo que tem mais de dois empregados, cuja média salarial é maior que 100, obter o código de cargo, seguido do nome do cargo, seguido da soma dos salários dos empregados do cargo.

b) Para cada cargo que tem mais de dois empregados, cuja média salarial, considerando todos os empregados do cargo, exceto os dois primeiros, é maior que 100, obter o código de cargo, seguido do nome do cargo, seguido da soma dos salários dos empregados do cargo.

c) Para cada empregado que tem mais de dois cargos, ambos com média salarial maior que 100, obter o código e cargo, seguido do nome do cargo, seguido da soma dos salários dos empregados do cargo.

d) Para cada cargo que tem mais de dois empregados, cuja média salarial é maior que 100, obter um grupo de linhas que contém, para cada empregado do cargo, o código de seu cargo, seguido do nome de seu cargo, seguido da soma dos salários dos empregados do cargo.

e) Para cada cargo que tem mais de 100 empregados, cuja média salarial é maior que 2, obter o nome do cargo, seguido da soma dos salários dos empregados do cargo.

5. Considere uma tabela criada pelo script de criação e carga, apresentada a seguir:

```
create table empregado(
        cdemp integer not null,
        nome varchar(6),
        fone varchar(10),
        primary key (cdemp)
);

insert into empregado (cdemp, nome, fone) values (1, 'Toni', '282677');
insert into empregado (cdemp, nome, fone) values (2, 'Joao', '282677');
insert into empregado (cdemp, nome, fone) values (3, 'Maria', '260088');
insert into empregado (cdemp, nome, fone) values (4, 'Jose', '174590');
insert into empregado (cdemp, nome)       values (5, 'Ana');
```

Sobre esta base de dados, foram realizadas 3 consultas diferentes, a saber:
I. select count(*) from empregado
II. select count(fone) from empregado
III. select count(*) from empregado where nome like "_o%"

As três consultas (I, II e III) retornarão, respectivamente, os valores:
a) 5,4,3.
b) 2,5,4.
c) 5,5,2.
d) 5,5,5.
e) 4,5,2.

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados**. 1. ed. São Paulo: Érica, 2014.

AMADEU, Cláudia (org). **Banco de Dados**. São Paulo: Pearson, 2014.

BARBOZA, Fabrício. **Modelagem e Desenvolvimento de Banco de Dados**. Porto Alegre: SAGAH, 2018.

CARDOSO, Virginia. CARDOSO, Giselle. **Sistemas de Banco de Dados: uma abordagem introdutória e prática**. São Paulo: Saraiva, 2012.

CARDOSO, Virginia; CARDOSO, Giselle. **Linguagem SQL**: **fundamentos e práticas**. São Paulo: Saraiva, 2013.

DATE, Chris. **Introdução a Sistemas de Banco de Dados**. 8.ed. São Paulo, Campus: 2004.

ELSMARI, Ramez. NAVATHE, Shamkant**. Sistemas de Banco de Dados**. 7. ed. São Paulo: Pearson, 2019.

HEUSER, Carlos. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

MACHADO, Felipe. Banco de Dados: Projeto e Implementação. 4. ed. São Paulo: Érica, 2014.

MEDEIROS, Luciano. **Banco de dados: princípios e práticas**. Curitiba: Intersaberes, 2013.

MANNINO, Michael V. **Projeto, desenvolvimento de aplicações e administração de banco de dados**. 3. ed. Porto Alegre: AMGH, 2014.

RAMARKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados**. 3. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de Software**, 9. Ed. São Paulo: Pearson Prentice-Hall, 2011.

TEOREY, Toby. et al. **Projeto e Modelagem de Banco de Dados**. 2. ed. São Paulo: Elsevier, 2014.

uniavan.edu.br