Banco de Dados Relacional

Prof. Neli Miglioli Sabadin

UNIASSELVI

Indaial – 2020

1ª Edição

Elaboração:
Prof. Neli Miglioli Sabadin

Revisão, Diagramação e Produção:
Centro Universitário Leonardo da Vinci – UNIASSELVI

Ficha catalográfica elaborada na fonte pela Biblioteca Dante Alighieri
UNIASSELVI – Indaial.

S113b

Sabadin, Neli Miglioli

Banco de dados relacional. / Neli Miglioli Sabadin. – Indaial: UNIASSELVI, 2020.

193 p.; il.

ISBN 978-65-5663-023-6

1. Banco de dados relacionais. - Brasil. Centro Universitário Leonardo Da Vinci.

CDD 005.756

Impresso por:

APRESENTAÇÃO

Existem vários bancos de dados e você já deve ter parado para pensar qual é a sua relação com eles, ou qual é a importância deles para a sociedade?

Atualmente muito se fala em dados, informações, leis de segurança de dados. Estamos vivendo na era da informação e precisamos conhecer os bancos de dados. A sociedade hoje vive em torno de dados. Suas preferências e necessidades de compra estão disponíveis na rede social, cada vez que você interagir com um amigo. Suas contas de telefone, luz, água, telefone e várias outras possuem dados relacionados que formam o valor a ser pago na fatura. Você consegue ver a média dos seus consumos e as empresas, com base nos seus dados, conseguem identificar discrepâncias em suas contas e até mesmo indicar outros serviços, com base no consumo. Eles possuem em suas bases de dados, além do seu cadastro e o consumo, e com isso conseguem saber muito mais do que o valor da sua fatura.

Pense que cada cadastro que você faz na internet, para acessar um determinado site, ou baixar um material, você está fazendo um cadastro e uma empresa está armazenando seus dados. Se você comprou algum produto em estabelecimento comercial, o seu CPF provavelmente existe na base de dados dele. Como vimos, existe uma grande "ligação" entre você e vários bancos de dados.

Isso pode tornar o conteúdo desta disciplina muito interessante. Neste livro didático serão abordados vários assuntos que vão desde as primeiras formas de armazenamento de dados, a evolução dos bancos de dados, as relações entre os dados, e como armazenar e recuperar essas informações em um banco de dados relacional. Além disso, aprenderá quais são os conceitos de bancos de dados relacional, como criar uma base de dados e até poderá programar nesta base de dados, com procedures e funções.

Aproveitamos a oportunidade para destacar a importância de desenvolver as autoatividades, lembrando que essas atividades NAO SÃO OPCIONAIS. E que objetivam a fixação dos conceitos apresentados. Em caso de dúvida na realização das atividades, sugerimos que você entre em contato com seu Tutor Externo ou com a tutoria da Uniasselvi, não prosseguindo as atividades sem ter sanado todas as dúvidas que surgirem.

Assim, temos muito para aprender! Bons estudos!

Prof. Neli Miglioli Sabadin

Você já me conhece das outras disciplinas? Não? É calouro? Enfim, tanto para você que está chegando agora à UNIASSELVI quanto para você que já é veterano, há novidades em nosso material.

Na Educação a Distância, o livro impresso, entregue a todos os acadêmicos desde 2005, é o material base da disciplina. A partir de 2017, nossos livros estão de visual novo, com um formato mais prático, que cabe na bolsa e facilita a leitura.

O conteúdo continua na íntegra, mas a estrutura interna foi aperfeiçoada com nova diagramação no texto, aproveitando ao máximo o espaço da página, o que também contribui para diminuir a extração de árvores para produção de folhas de papel, por exemplo.

Assim, a UNIASSELVI, preocupando-se com o impacto de nossas ações sobre o ambiente, apresenta também este livro no formato digital. Assim, você, acadêmico, tem a possibilidade de estudá-lo com versatilidade nas telas do celular, tablet ou computador.

Eu mesmo, UNI, ganhei um novo layout, você me verá frequentemente e surgirei para apresentar dicas de vídeos e outras fontes de conhecimento que complementam o assunto em questão.

Todos esses ajustes foram pensados a partir de relatos que recebemos nas pesquisas institucionais sobre os materiais impressos, para que você, nossa maior prioridade, possa continuar seus estudos com um material de qualidade.

Aproveito o momento para convidá-lo para um bate-papo sobre o Exame Nacional de Desempenho de Estudantes – ENADE.

Bons estudos!

Olá acadêmico! Para melhorar a qualidade dos materiais ofertados a você e dinamizar ainda mais os seus estudos, a Uniasselvi disponibiliza materiais que possuem o código *QR Code*, que é um código que permite que você acesse um conteúdo interativo relacionado ao tema que você está estudando. Para utilizar essa ferramenta, acesse as lojas de aplicativos e baixe um leitor de *QR Code*. Depois, é só aproveitar mais essa facilidade para aprimorar seus estudos!

BATE PAPO SOBRE O ENADE!

Olá, acadêmico!

Você já ouviu falar sobre o **ENADE**?

Se ainda não ouviu falar nada sobre o ENADE, agora você receberá algumas informações sobre o tema.

Ouviu falar? Ótimo, este informativo reforçará o que você já sabe e poderá lhe trazer novidades.

Vamos lá!

Qual é o significado da expressão ENADE?

EXAME NACIONAL DE DESEMPENHO DOS ESTUDANTES

Em algum momento de sua vida acadêmica você precisará fazer a prova ENADE.

Que prova é essa?

É **obrigatória**, organizada pelo INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira.

Quem determina que esta prova é obrigatória... O **MEC – Ministério da Educação**.

O objetivo do MEC com esta prova é o de avaliar seu desempenho acadêmico assim como a qualidade do seu curso.

Fique atento! Quem não participa da prova fica impedido de se formar e não pode retirar o diploma de conclusão do curso até regularizar sua situação junto ao MEC.

Não se preocupe porque a partir de hoje nós estaremos auxiliando você nesta caminhada.

Você receberá outros informativos como este, complementando as orientações e esclarecendo suas dúvidas.

Você tem uma trilha de aprendizagem do ENADE, receberá e-mails, SMS, seu tutor e os profissionais do polo também estarão orientados.

Participará de webconferências entre outras tantas atividades para que esteja preparado para #mandar bem na prova ENADE.

Nós aqui no NEAD e também a equipe no polo estamos com você para vencermos este desafio.

Conte sempre com a gente, para juntos mandarmos bem no ENADE!

UNIASSELVI

Olá, acadêmico! Iniciamos agora mais uma disciplina e com ela um novo conhecimento.

Com o objetivo de enriquecer seu conhecimento, construímos, além do livro que está em suas mãos, uma rica trilha de aprendizagem, por meio dela você terá contato com o vídeo da disciplina, o objeto de aprendizagem, materiais complementares, entre outros, todos pensados e construídos na intenção de auxiliar seu crescimento.

Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

Conte conosco, estaremos juntos nesta caminhada!

SUMÁRIO

UNIDADE 1

FUNDAMENTOS DE BANCO DE DADOS

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você deverá ser capaz de:

- compreender a necessidade de um banco de dados;
- conhecer a evolução dos bancos de dados;
- conhecer as formas de representar um banco de dados, através dos modelos conceitual, lógico e físico;
- conhecer as características de um banco de dados relacional.

PLANO DE ESTUDOS

Esta unidade está dividida em três tópicos e em cada um deles, no decorrer da unidade, você encontrará atividades, com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – INTRODUÇÃO A BANCO DE DADOS

TÓPICO 2 – MODELOS DE REPRESENTAÇÃO EM BANCOS DE DADOS

TÓPICO 3 – ENTIDADES, ATRIBUTOS E RELACIONAMENTOS

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

TÓPICO 1


INTRODUÇÃO A BANCO DE DADOS

1 INTRODUÇÃO

Desde os primórdios, os seres humanos tinham a necessidade de armazenar informações, como, por exemplo, o registro de desenhos nas paredes. Mesmo sem perceber, estamos sempre armazenando e fazendo uso de um banco de dados, não necessariamente informatizado. Como o armazenamento das atividades através das antigas agendas, dos contatos telefônicos e da caixa de fotografias antigas.

Semelhante à evolução dos bancos de dados, também evoluímos e armazenamos as informações de outras maneiras, as agendas e a listas telefônicas hoje estão juntas no celular.

Seguindo essa mesma evolução, os bancos de dados são sistemas que foram desenvolvidos com o objetivo de facilitar o trabalho, além de garantir eficiência, rapidez e segurança na recuperação das informações.

2 NECESSIDADE DE ARMAZENAMENTO

Você sabe a quantidade de sites em que já fez um cadastro? Quantas senhas você já cadastrou? O número de telefone dos seus amigos? A data de aniversário dos seus parentes e colegas de trabalho? A receita daquele bolo que você gostou? Ou já revelou todas as fotos que tirou? Indo um pouco mais além, e menos pessoal, você já parou para pensar quantos dados você recebe durante um dia, quantos deles usa como informação e ainda, quantos deles se transformam em conhecimento?

Segundo Santos (s.d.), guardar informações é uma tarefa antiga e que sempre foi realizada pelo ser humano, seja ela através da escrita, fala ou até mesmo através das lendas e histórias que foram passadas de geração em geração. A busca por manter as informações para as gerações futuras era comum nas civilizações mais antigas, que buscavam sempre um meio de preservar as suas culturas e dar continuidade aos seus costumes.

Graças à evolução constante na esfera industrial, social, intelectual e tecnológica, os métodos de armazenamento e transmissão de dados e informações passaram constantemente por evoluções, sempre os tornando mais eficientes e rápidos. Vamos começar contextualizando o que é cada um deles.

3 DADO, INFORMAÇÃO E CONHECIMENTO

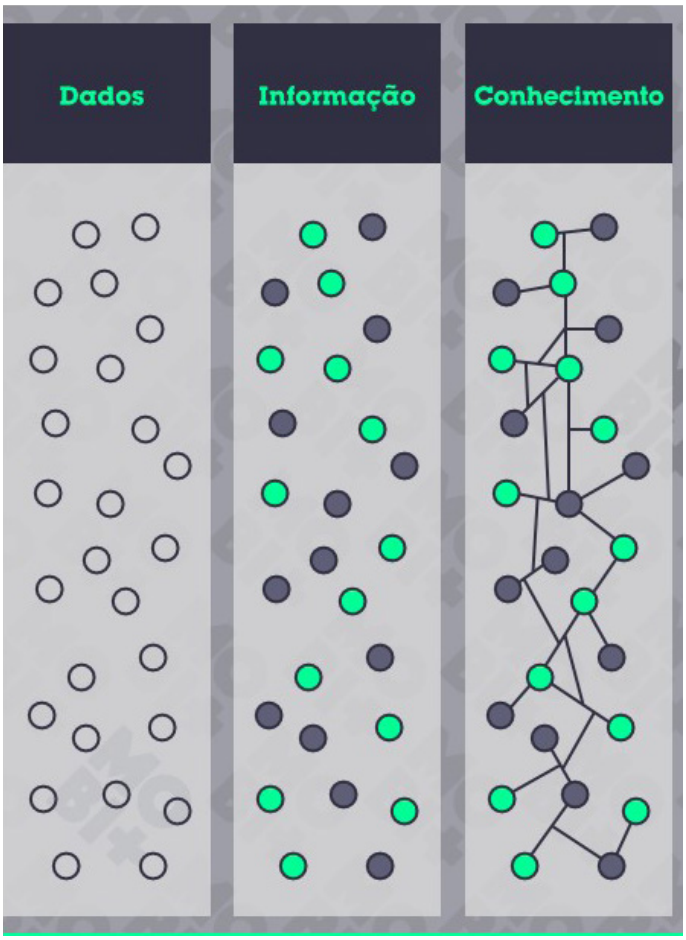
A importância desse tópico deve-se que a informação é um conceito central na área de sistemas de informação, pois ela é o recurso mais valioso e importante nas organizações na sociedade atual, que também é conhecida como sociedade da informação.

Apesar do uso muitas vezes equivocado desses três termos, precisamos ter muito claro e compreender a diferença que existe entre dado, informação e conhecimento.

- Dado: é um conteúdo que ainda não foi processado para gerar um significado. Pode-se dizer que dado é a menor unidade de conteúdo que tem significado no mundo real. Por exemplo, um dado seria o dia 25/12.
- Informação: é qualquer fato ou conhecimento do mundo real e que pode ou não ser registrado/armazenado. Eu saber que no dia 25/12 é comemorado o Natal, ou que é feriado.
- Conhecimento: Após ter informação que no dia 25/12 é Natal, o conhecimento se aplica em trabalhar essa informação. Ele é um processamento subjetivo da informação e individual. Ou seja, a pessoa pode querer comprar presentes, ou que irá reencontrar parentes que não via há muito tempo. O conhecimento se baseia em experiências anteriores.

Observe a Figura 1, ela representa nosso dia a dia, quando recebemos milhares e milhares de dados, muitos sem relevância alguma, outros nos trazem informações, mas somente alguns dados irão se transformar em conhecimento.

FIGURA 1 – DADO, INFORMAÇÃO E CONHECIMENTO



FONTE: <https://bit.ly/2wF7zX8>. Acesso em: 7 jul. 2019.

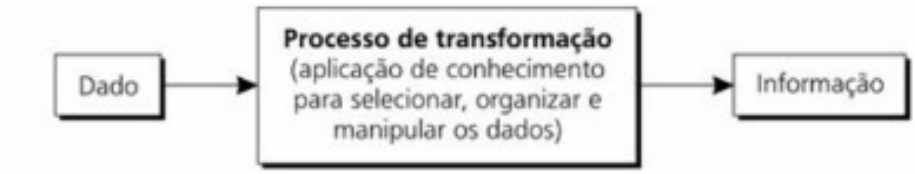
Podemos dizer que um dado é como uma gota d´água, que quando você está caminhando na calçada e sente caindo em você. Isoladamente este fato não representa que está chovendo, pois pode ter sido a gota de um ar condicionado, um pássaro, melhor continuar com a opção do ar condicionado. Já a informação é o passo seguinte, é a percepção de que vai chover, pois você sente um vento mais forte, as nuvens estão com cores mais acinzentadas e a quantidade de pingos começa a aumentar. Então, com base nesses fatos, você chega à conclusão que realmente vai chover. Mas isso não basta, pois você ainda vai se molhar, por isso é preciso o conhecimento para interagir com a nova informação, de que você vai se molhar. Quando você entende isso, consegue transformar a gama de dados em informação e gerar isso em uma ação, isso é o conhecimento. Pois a partir das informações que tem, pode prever o que vai acontecer, seja pelo histórico dos fatos anteriores ou por novas conclusões, está utilizando o seu conhecimento.

Audy (2007) define dado como um fato bruto, por exemplo, o nome de um cliente ou funcionário, número de matrícula de um aluno, código de um produto etc., ou suas representações, por exemplo, imagens, sons, números etc. Vale res-

saltar que podem ou não ser úteis ou pertinentes para um processo em particular. São vários os tipos de dados que podem ser utilizados para representar esses fatos. Por exemplo: alfanumérico, que são números, letras ou outros caracteres. O tipo imagem, que como o nome diz, são imagens ou figuras, além de áudios e vídeos.

Já a informação, ainda segundo Audy (2007), é uma coleção de fatos organizados de forma a possuir um valor adicional aos fatos em si. Em resumo, são dados concatenados, que passaram por um processo de transformação, cuja forma e conteúdo são apropriados determinado cenário.

FIGURA 2 – DADO E INFORMAÇÃO



FONTE: Audy (2007, p. 95)

Faz-se necessária uma atenção especial na definição desse tema, pois os cenários vão garantir a qualidade dos dados e das informações obtidas no banco de dados. Pois a informação possui uma série de características que determinam seu valor para a organização ou processo em análise. Podemos identificar as principais características da informação, segundo Audy (2007, p. 95):

- Precisa: sem erros; em alguns casos, informações incorretas são geradas porque dados incorretos são lançados como entrada no processo de transformação (entra lixo, sai lixo).
- Completa: contém todos os fatos relevantes no processo em análise.
- Econômica: deve ser relativamente econômica para ser gerada, pois os tomadores de decisão deverão balancear o valor da informação com o custo para ser obtida.
- Flexível: deve estar armazenada de forma a ser utilizada de formas diferentes e para apoiar processos diferentes.
- Confiável: é dependente da confiabilidade dos dados de origem e dos métodos de coleta de dados.
- Relevante: são importantes para os tomadores de decisão decidirem sobre um determinado processo ou decisão.
- Clara (simples): deve ser simples; normalmente informações detalhadas e
- Complexas: não são úteis aos tomadores de decisão, bem como devem estar filtradas em quantidades compatíveis com as necessidades e as capacidades de processamento do tomador de decisão.
- Veloz: é entregue quando necessária, nem antes, nem depois.
- Verificável: deve permitir uma verificação por parte do tomador de decisão, quando necessário.
- Acessível: deve ser facilmente acessível por usuários autorizados, no formato adequado e no momento certo.
- Segura: segurança de acesso somente por pessoas autorizadas.

Ainda segundo Audy (2007), a origem das informações pode ser formal ou informal. Também podem ser obtidas no contexto organizacional (interno) ou no meio ambiente onde a organização está inserida (externo), como detalharemos mais adiante na fase de modelagem.

Tendo conhecimento da importância dos dados e das informações no contexto de um banco de dados vamos conhecer como surgiram os bancos de dados, pois a vontade do ser humano em registrar os dados para uso futuro não é de hoje, vamos rever e conhecer mais sobre a origem e os tipos de banco de dados.

4 O SURGIMENTO DOS BANCOS DE DADOS

Desde os primórdios dos tempos, o homem sempre teve a necessidade de deixar registrados os principais eventos, e informações mais importantes que porventura pudessem ser utilizadas futuramente, mesmo sem ter uma linguagemespecífica para isso, utilizando-se de técnicas de pinturas pré-históricas, as inscriçõeshieroglíficas dos egípcios. Tudo com um único objetivo, registrar dados.

A evolução do armazenamento de dados teve em suas origens dois personagens muito importantes, conheça a Johannes Gutenberg e Herman Hollerith.

Graças às técnicas de impressão de Gutenberg, detalhadas por Alves (2014), a informação poderia ser replicada com mais facilidade, devido à tecnologia de impressão e tipografia. Dando um salto já na era dos computadores, no início da década de 1960, foram lançados os primeiros sistemas gerenciadores de banco de dados (SGBD). Que tinham como objetivo o aumento na produtividade nas atividades de desenvolvimento e manutenção de sistemas. Todos esses avanços, a possibilidade de gerar mais e mais cópias dos dados também gerava outro problema, a duplicidade das informações. Agora os dados estavam impressos ou salvos nos novos computadores, mas como garantir sua integridade e a sua segurança. Foi aí que surgiu a necessidade dos bancos de dados.

Podemos dizer que os programas de banco de dados talvez sejam os mais antigos que foram desenvolvidos, pois anterior a eles, os programas trabalhavam com arquivos sequenciais, que eram os únicos recursos para a gravação e leitura dos

dados. Obviamente, isso gerava muitos transtornos, como a ausência de controle de acesso para mais de um usuário, a impossibilidade de execução de mais de um processo no mesmo arquivo. Sem contar que a definição da estrutura do arquivo, era feita no próprio programa. Além de problemas como os definidos por Sanches (2005):

- **Redundância e inconsistência de dados**: muitos programadores diferentes e programas implementados em linguagens diferentes podem gerar arquivos de formatos diferentes. Informações podem estar duplicadas em diversos lugares. Geram inconsistência, pois estas cópias podem estar com valores diferentes.
- **Dificuldade no acesso aos dados**: um diretor deseja a lista de todos os clientes que moram na cidade de CEP 34863. Ou ele extrai manualmente esta informação de uma lista de clientes ou pede a um programador escrever um programa que aplicativo. Suponha mais tarde que o mesmo diretor deseja uma lista com os clientes com mais de $10000. Tal lista não existe e novamente o diretor tem as duas opções.
- **Isolamento dos dados**: como os dados estão espalhados, em arquivos separados e com formatos diferentes, é difícil escrever novos programas aplicativos para recuperar os dados adequados.
- **Anomalias de acesso concorrente**: para aperfeiçoar o desempenho geral do sistema e obter tempo de reposta mais rápido, deixamos que múltiplos clientes acessem e atualizem os dados simultaneamente. Isso gera dados inconsistentes. Exemplo: dois clientes sacarem dinheiro de uma mesma conta corrente.
- **Problemas de segurança**: nem todo usuário do sistema de BD deve ter acesso a todos os dados. Por exemplo: o RH pode ter acesso às informações cadastrais dos clientes, mas não aos valores de conta corrente. Se novos programas aplicativos forem adicionados, é difícil assegurar tais restrições de segurança.
- **Problemas de integridade**: os valores dos dados armazenados necessitam satisfazer certas restrições. Por exemplo, o saldo nunca estar abaixo de $25. Estas restrições podem estar contidas nos programas aplicativos, mas quando novas restrições forem adicionadas, é difícil de alterar estes programas.

Para resolver esses problemas apresentados foram surgindo alguns tipos de banco de dados, que tinham suas limitações de acordo com a capacidade do hardware, mas que foram evoluindo com o passar dos anos e se adequando às necessidades específicas de determinadas linguagens.

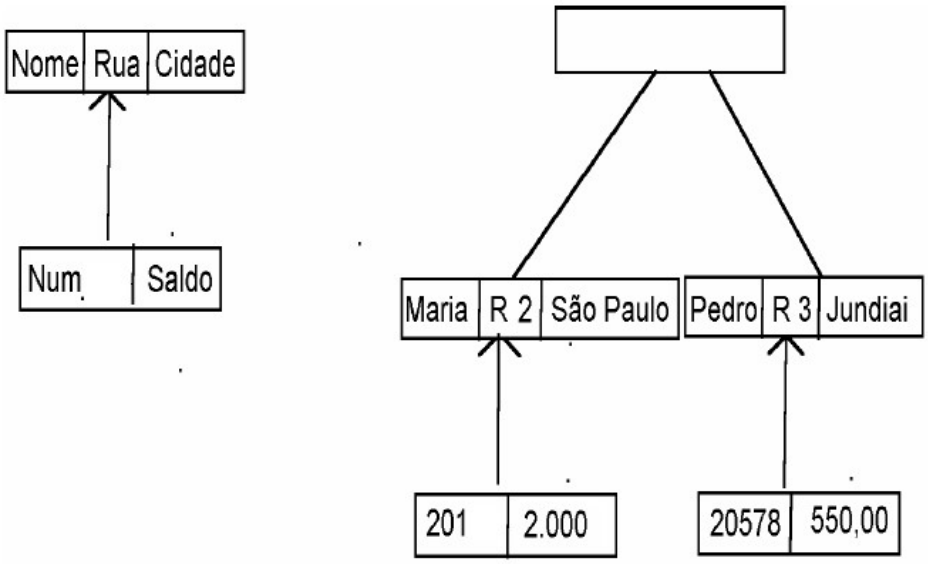
5 TIPOS DE BANCOS DE DADOS

Nesta seção vamos listar os modelos mais conhecidos, e apenas de maneira introdutória, uma vez que o objetivo deste material são os bancos de dados relacionais. A ordem dos bancos apresentados nesse tópico segue a sua data de criação.

5.1 BANCO DE DADOS HIERÁRQUICO

Nascido na década de 1960, esse tipo de banco de dados, é um dos mais antigos métodos de organização e armazenamento de dados, segundo Lehmkuhl (2013). Ainda segundo o autor, toda a estrutura está organizada no formato de uma árvore, onde cada registro filho, poderá ter apenas um registro pai, e cada registro pai, poderá ter vários registros filhos.

FIGURA 3 – MODELO DE DADOS HIERÁRQUICO



FONTE: O autor

Reforçado esse conceito, Gomes (2012) afirma que um banco de dados hierárquico é organizado em forma de pirâmide estendendo-se para baixo. Áreas afins ou registros são agrupados de modo que registros de nível não são mais elevados que outros registros inferiores. Ou seja, no topo da pirâmide ou o registo principal é chamado registro raiz. Vale lembrar que não há um registro filho, sem estar atrelado a um registro pai, mas o inverso pode ser verdade, onde um registro pai, não necessariamente terá um registro filho. Eles trabalham movendo de cima para baixo, onde a pesquisa é realizada começando pelo topo da árvore e indo para baixo, passando de pai para filho até que o registro da criança apropriada seja encontrado. Fazendo uma analogia com uma árvore genealógica familiar, em que as pessoas mais velhas como avós, depois seguido pelos filhos dos avós e depois os filhos dos filhos dos avós.

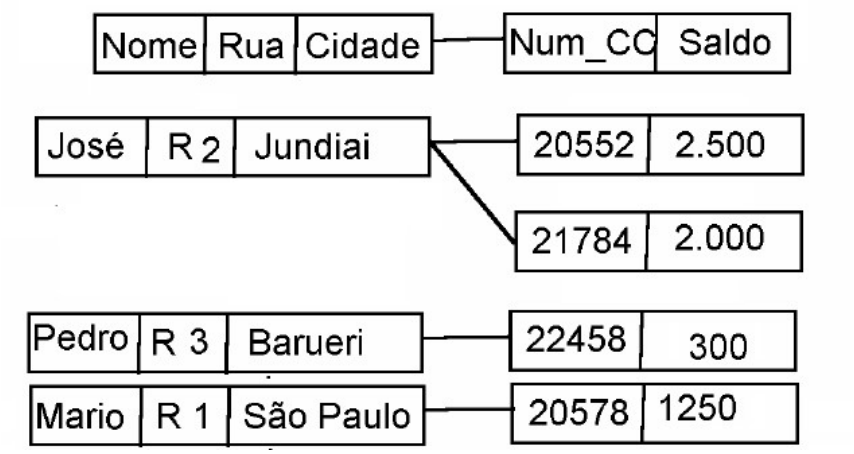
Como vantagem Gomes (2012) afirma que esse tipo de banco de dados hierárquicos pode ser acessado e atualizado rapidamente porque nessa estrutura as relações entre os registos são previamente definidas. Mas, como nem tudo são flores, temos também desvantagens, onde que cada criança na árvore pode ter apenas um pai, e os relacionamentos ou ligações entre as crianças não são permi-

tidas, mesmo se elas fazem sentido do ponto de vista lógico. Os bancos de dados hierárquicos são tão rígidos em seu projeto que a adição de um novo campo ou registro requer que o banco de dados inteiro seja redefinido.

5.2 BANCO DE DADOS EM REDE

Tendo como base o modelo hierárquico, Gomes (2012) afirma que o modelo em rede permite onde possuem a estrutura hierárquica, com algumas diferenças fundamentais. Onde ao invés de olhar como uma árvore de cabeça para baixo, ele se assemelha mais com uma teia de aranha. Os registros, filhos ou as crianças, agora são chamados de membros e os pais de proprietários. Como diferença mais significativa temos que cada criança ou membro pode ter mais de um pai (ou dono). Semelhante ao banco de dados hierárquico, neste também um limite para o número de ligações que podem ser feitas entre os registros.

FIGURA 4 – MODELO DE DADOS EM REDE



FONTE: O autor

Apresentando os modelos hierárquicos e em redes, vamos conhecer o tema do nosso livro de estudos, o modelo de dados relacional.

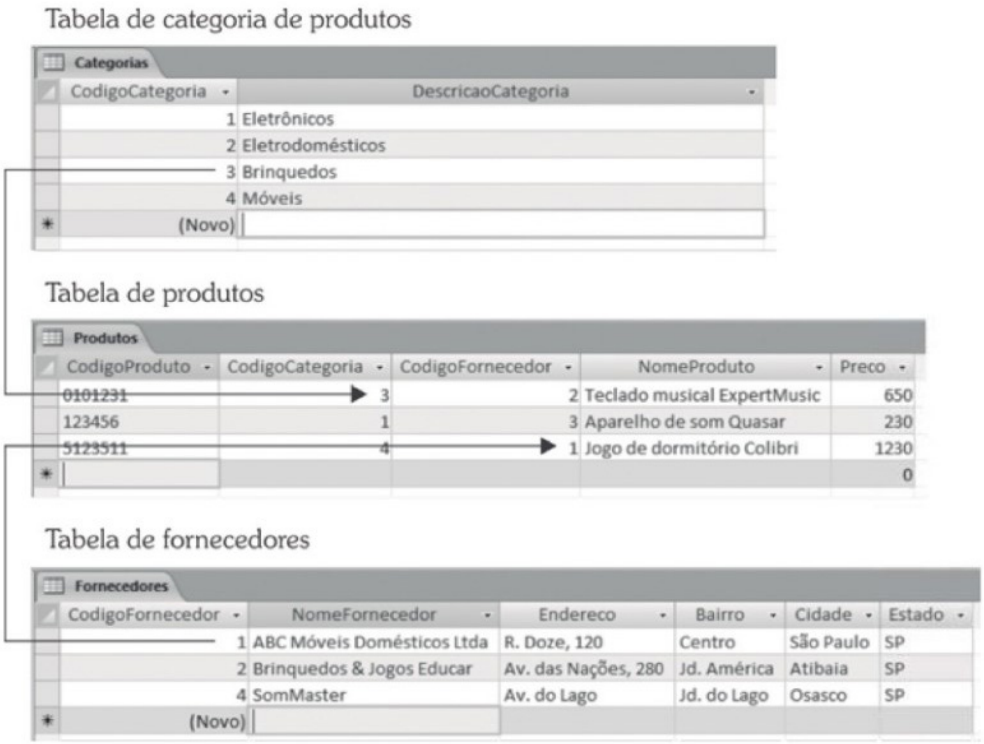
5.3 MODELO DE DADOS RELACIONAL

Segundo Lehmkuhl (2013), o modelo de dados relacional surgiu na década de 1970, como novo conceito em banco de dados. Esse modelo é largamente utilizado até os dias de hoje por aplicações de processamento de dados comerciais mais padronizadas. Ele elimina a complexidade de navegação ou busca de dados presentes nos hierárquicos e nos de rede.

Para Alves (2014) um banco de dados relacional é caracterizado por organizar os dados em tabelas (ou relações), formadas por linhas e colunas. O autor reforça:

Da mesma forma que na matemática, podem ser efetuadas operações entre dois ou mais conjuntos, como, por exemplo, obter os elementos que são comuns a ambos os conjuntos (tabelas/relações) num banco de dados relacional. Também é possível executar certas operações com essas tabelas, como ligar duas ou mais por meio de campos comuns em ambas. Quando uma operação de consulta é executada, o resultado é um conjunto de registros que pode ser tratado como uma tabela virtual (que só existe enquanto a consulta está ativa) (ALVES, 2014, p. 19).

FIGURA 5 – MODELO RELACIONAL



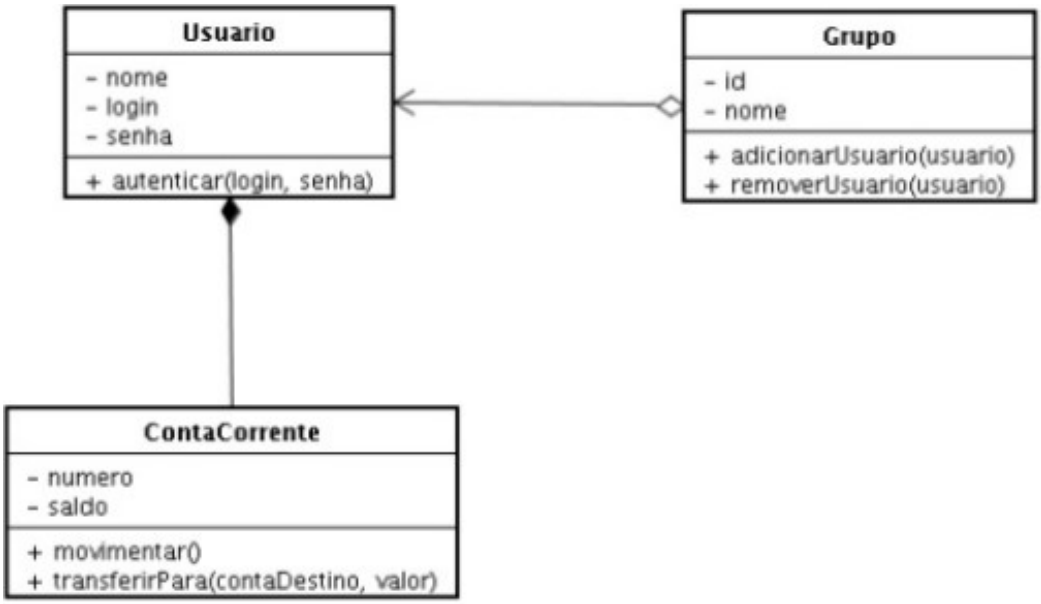
FONTE: Alves (2014, p. 21)

Para que um banco de dados seja considerado relacional, Alves (2014) afirma que ele deve seguir as doze regras de Codd (serão detalhadas na próxima seção). E que as informações estejam divididas em tabelas e que certas informações de uma tabela são obtidas a partir de outras tabelas. E que pode ser necessário, campos em comuns entre as tabelas, para definir o relacionamento entre elas. Vejamos o exemplo, onde o código da categoria (tabela categoria de produto) aparece na tabela de produto, que também recebe da tabela de fornecedores, o código do fornecedor. Veremos mais detalhadamente esses relacionamentos.

5.4 BANCO DE DADOS ORIENTADO A OBJETOS

Segundo Alves (2014), esse tipo de banco de dados surgiu em meados de 1980, em virtude da necessidade de armazenamento de dados que não era possível com os sistemas relacionais (que veremos a seguir) devido aos seus limites, como os sistemas de geoprocessamento (Sistemas de Informações Geográficas (SIG)) e CAD/CAM/CAE, que são baseados em tipos de dados complexos.

FIGURA 6 – MODELO ORIENTADO A OBJETOS



FONTE: O autor

Neste modelo, segundo Lehmkuhl (2013), o banco de dados é definido como uma coleção de objetos, ou elementos de software reutilizáveis, com recursos e métodos associados. Graças a essa estruturação, ela permite uma adaptação menos dolorosa, porque vários programadores já estão familiarizados em linguagens orientadas a objeto, o que em termos de conceito são iguais a este tipo de banco. Onde tudo gira em torno de objetos, que possuem atributos e propriedades com mais uma gama de recursos para atingir seus objetivos.

5.5 BANCO DE DADOS NOSQL

O termo NoSQL, segundo Devmedia (2012), começou a ser utilizado em 1998 a partir de uma solução que não oferecia uma interface SQL, procurando se livrar de certas regras e estruturas que norteiam o Modelo Relaciona. Ele busca atenderprincipalmente às deficiências referentesà escalabilidade, performance e disponibilidade promovendo uma alternativa de alto armazenamento com velocidade e grande disponibilidade.

A proposta dos bancos NoSQL na realidade não é extinguir o Modelo Relacional, mas utilizá-lo em casos onde é necessária uma maior flexibilidade na estruturação do banco. A principal diferença entre banco de dados relacionais e não relacionais está na modelagem dos dados. Conforme o comparativo apresentado pelo Devmedia (2012).

FIGURA 7 – COMPARATIVO NOSQL E RELACIONAL

	Banco de Dados Relacional	Banco de Dados NoSQL
Escalonamento	É importante lembrar que é possível ser feito o escalonamento em um Modelo Relacional, no entanto, é muito complexo. Possui uma natureza estruturada, portanto, a inserção dinâmica e transparente de novos nós a tabela não é realizada naturalmente.	Não possui um esquema pré-definido fazendo com que este tipo de modelo seja flexível o que favorece a inserção transparente de outros elementos.
Consistência	Neste quesito, o Modelo Relacional se mostra forte. As suas regras de consistência são bastante rigorosas no que diz respeito à consistência das informações.	É realizada eventualmente no modelo: tem apenas a garantia que se não houver nenhuma atualização nos dados, todos os acessos aos itens devolverão o último valor que foi atualizado.
Disponibilidade	Por não conseguir trabalhar de forma eficiente com a distribuição de dados, o Modelo Relacional acaba não suportando uma demanda muito grande de informações.	Outro ponto forte neste modelo é o que diz respeito à disponibilidade, pois possui um alto nível de distribuição de dados, permitindo assim que seja possível fazer com que um enorme fluxo de solicitações aos dados seja atendido com a vantagem do sistema ficar indisponível o menor tempo possível.

FONTE: <https://bit.ly/3ak41bE>. Acesso em: 7 jul. 2019.

Conhecer o novo paradigma que vem surgindo na área de banco de dados é interessante para aqueles que trabalham com banco de dados. Leia o artigo disponível em: https://www.devmedia.com.br/introducao-aos-bancos-de-dados-nosql/26044.

5.6 SISTEMAS GERENCIADORES DE BANCO DE DADOS–SGBD

Quando falamos de um sistema de gerenciamento de banco de dados, estamos nos referindo a um conjunto de programas de software. Com ele, o usuário consegue criar, editar, atualizar, armazenar e recuperar dados em tabelas de banco de dados. Segundo Lehmkuhl (2013, p. 11), "Um Sistema Gerenciador de Banco de Dados é um software responsável pelo gerenciamento de base de dados. Uma das principais características dele é retirar da aplicação à estruturação dos dados, deixando de forma transparente o acesso aos mesmos".

E como principal vantagem no uso de um sistema de gerenciamento de banco de dados é o estado coerente dos dados que se encontram armazenados no banco. Garantindo que informações extraídas sejam confiáveis e de grande credibilidade.

Vale ressaltar ainda que os bancos de dados relacionais seguem o modelo ACID, (que é um acrônimo de **atomicidade, consistência, isolamento e durabilidade**) para preservar a integridade de uma transação. Este conjunto de procedimentos é dividido em quatro propriedades, e são elas: segundo Devmedia (2012).

- **Atomicidade**: as ações que compõem a ação da transação devem ser concluídas com sucesso para ser efetivada. Se esta transação falhar, será feito o rollback.
- **Consistência**: todas as regras/restrições descritas no banco de dados devem ser obedecidas garantindo que o banco de dados passe de uma forma consistente para outra forma consistente.
- **Isolamento**: neste caso, a propriedade de isolamento garante que a transição não será interferida por nenhuma outra transação concorrente.
- **Durabilidade**: os resultados de uma transação são permanentes, ou seja, o que foi salvo não será mais perdido.

Todos esses diferentes recursos auxiliaram a manter os SGBDs Relacionais sempre em uma posição de predominância entre os mais diversos tipos de ambientes computacionais. Heuser (2009) afirma quea integridade de dados é um dos objetivos primordiais de um SGBD. Caso esses dados estejam íntegros, é o mesmo que dizer que eles refletem corretamente a realidade representada pelo banco de dados e que são consistentes. A restrição de integridade é uma maneira de garantir a que os SGBD oferecem para tentar garantir a integridade. Uma restrição de integridade é uma regra de consistência de dados que é garantida pelo próprio SGBD. Heuser (2009, p. 91) afirma que no caso da abordagem relacional, costuma-se classificar as restrições de integridade nas seguintes categorias:

- Integridade de domínio: restrições deste tipo especificam que o valor de um campo deve obedecer a definição de valores admitidos para a coluna (o domínio da coluna). Nos SGBD relacionais comerciais, é possível usar apenas domínios pré-definidos (número inteiro, número real, alfanumérico de tamanho definido, data etc.). O usuário do SGBD não pode definir domínios próprios de sua aplicação (por exemplo, o domínio dos dias da semana ou das unidades da federação).

- Integridade de vazio: através deste tipo de restrição de integridade é especificado se os campos de uma coluna podem ou não ser vazios (se a coluna é obrigatória ou opcional). Como já foi mencionado, campos que compõem a chave primária sempre devem ser diferentes de vazio.
- Integridade de chave: trata-se da restrição que define que os valores da chave primária e alternativa devem ser únicos.
- Integridade referencial: é a restrição que define que os valores dos campos que aparecem em uma chave estrangeira devem aparecer na chave primária da tabela referenciada.

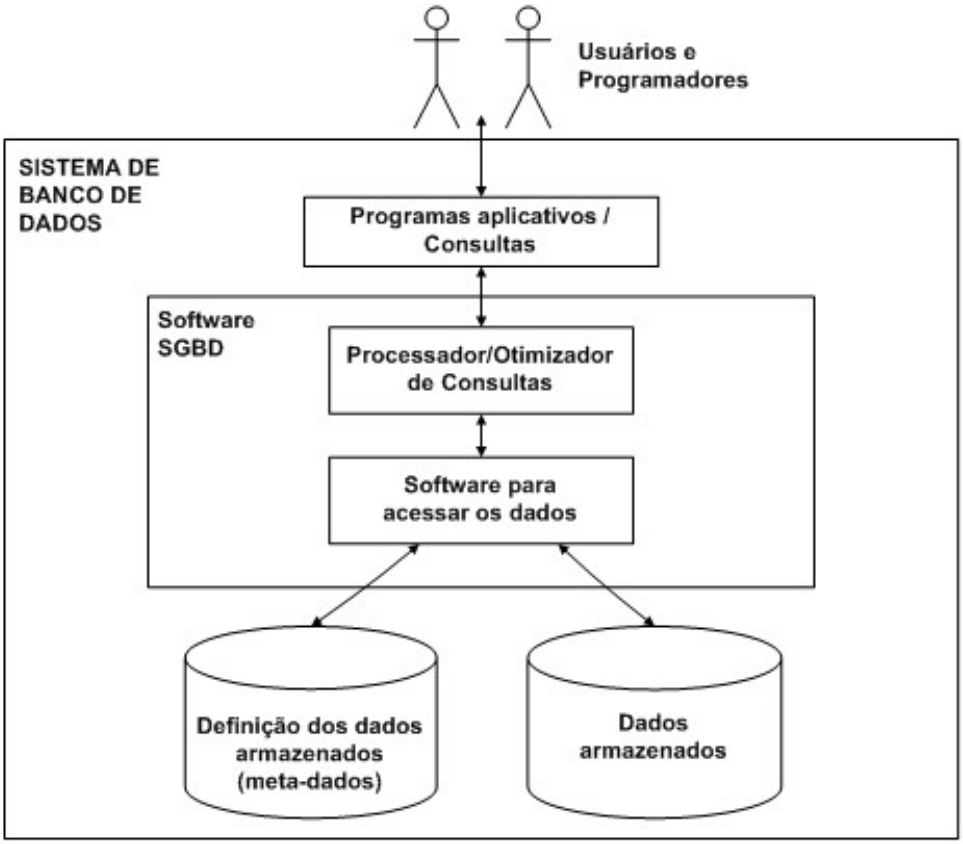
Ainda segundo Heuser (2009), ele afirma que essas restrições, não são preocupações dos desenvolvedores, mas que devem ser garantidas automaticamente por um SGBD relacional.

5.7 CAMADAS DE UM SGBD

Como podemos observar na figura a seguir, os SGBD, é formado por módulos com funcionalidades bem claras. Onde cada módulo tem sua responsabilidade no processo de gerenciamento dos dados. Usuários e programadores interagem com esses módulos a fim de obter seus resultados. Vejamos como são definidas segundo LEHMKUHL (2013):

- **Programas aplicativos/consultas**: essa é a camada que se relaciona diretamente com o usuário comum. Ele utiliza normalmente um sistema e é através desse sistema que o acesso ao banco de dados é feito. Já um programador tem ferramentas que são específicas para trabalhar com a tecnologia que o SGBD disponibiliza. Essas ferramentas tem um foco mais centrado na visualização de dados e manutenção de suas estruturas.
- **Processador/otimizador de consultas**: aqui ocorre a interpretação detodos os acessos que são feitos na base de dados com um foco maior em otimização. Ou seja, ele é responsável por processar e definir "caminhos" para que o usuário tenha uma resposta a sua solicitação com performance.
- **Software para acessar os dados**: é o modulo responsável por recuperar os dados do local onde eles estão armazenados. Esses dados são divididos em dois grupos: os dados armazenados e a definição dos dados armazenados. O primeiro grupo se refere aos dados que um usuário comum manipula, como por exemplo, nomes de pessoas, cidades, endereços, telefones etc. Já o segundo define a forma de organização dos dados do primeiro grupo. É composto por tabelas, índices, relacionamentos etc.

FIGURA 8 - DIAGRAMA SIMPLIFICADO DA ARQUITETURA DO SISTEMA DE BANCO DE DADOS



FONTE: <https://www.ime.usp.br/~andrers/aulas/bd2005-1/aula5.html>. Acesso em: 7 jul. 2019.

Como vimos, o gerenciador de banco de dados é um módulo de um programa que provê a interface entre os dados de baixo nível armazenados num banco de dados e os programas de aplicação e as solicitações submetidas ao sistema. O gerenciador de banco de dados é responsável pelas seguintes tarefas, segundo Sanches (2005):

- **Interação com o gerenciamento de arquivos**: os dados não trabalhados são armazenados no disco usando o sistema gerenciador de arquivos, que geralmente é fornecido por algum sistema operacional. O gerenciador de banco de dados traduz os diversos comandos da DML em comandos de baixo nível do gerenciador de arquivos. Portanto, o gerenciador de banco de dados é responsável pelo armazenamento, pela busca e pela atualização de dados no banco de dados.
- **Cumprimento de integridade**: os valores de dados armazenados num banco de dados precisam satisfazer certos tipos de restrições de consistência. Por exemplo, o número de horas que um empregado pode trabalhar em uma semana não pode ultrapassar um limite estabelecido (digamos, 80 horas). Tal restrição precisa ser especificada explicitamente pelo administrador do banco de dados. O sistema gerenciador de banco de dados pode então determinar se as atualizações no banco de dados resultam numa violação da restrição; em caso positivo, ações apropriadas precisam ser tomadas.

- **Cumprimento de segurança**: como discutido anteriormente, nem todo usuário do banco de dados necessita ter acesso a todo o banco de dados, para impor requisitos de segurança de acesso aos dados.
- **Cópias de reserva (backup) e recuperação (restore)**: um computador, como qualquer outro dispositivo mecânico ou elétrico, está sujeito a falhas. As causas das falhas incluem quebras de disco, falhas na energia e erros de software. Em cada um dos casos, as informações que se referem ao banco de dados podem ser perdidas. É responsabilidade do sistema gerenciador do banco de dados detectar tais falhas e restabelecer o estado do banco de dados como estava antes da ocorrência da falha. Isto é feito normalmente através da ativação de diversos procedimentos de recuperação e de cópias de reserva.
- **Controle de concorrência**: se diversos usuários atualizam o banco de dados concorrentemente, a consistência dos dados pode não ser mais preservada. Controlar a interação entre usuários simultâneos é outra atribuição do gerenciador de banco de dados.

Alves (2014) reforça que temos que lembrar que um Sistema de Gerenciamento de Banco de Dados (SGBD) é composto de uma coleção de arquivos inter-relacionados e de um conjunto de programas que permitem aos usuários fazer o acesso a estes arquivos e modificar os mesmos. E tem por incumbência básica manter os dados estáveis, ou seja, devem cuidar de toda a manutenção e atualização dos registros. A construção de rotinas, criação de interface ou desenvolvimento de aplicativos completos deve ficar a cargo de uma ferramenta destinada a essa tarefa, como uma linguagem de programação, que se comunica com o gerenciador utilizando um mecanismo oferecido por ele, como drivers de comunicação similares ao ODBC (Open Database Connectivity), ADO (ActiveX Data Objects) ou JDBC (Java Database Connectivity). Além das vantagens apresentas anteriormente, Ramakrishnan (2011, p. 7) sinaliza também:

- Independência de Dados: os programas aplicativos não devem, idealmente, ser expostos aos detalhes de representação e armazenamento de dados. O SGBD provê uma visão abstrata dos dados que oculta tais detalhes.
- Acesso Eficiente aos Dados: um SGBD utiliza uma variedade de técnicas sofisticadas para armazenar e recuperar dados eficientemente. Este recurso é especial- mente importante se os dados são armazenados em dispositivos de armazenamento externos.
- Integridade e Segurança dos Dados: se os dados são sempre acessados através do SGBD, ele pode forçar restrições de integridade. Por exemplo, antes de inserir informações sobre o salário de um funcionário, o SGBD pode verificar se o orçamento do departamento não está se excedendo. Além disso, ele pode forçar controles de acesso que governam quais dados estão visíveis a diferentes classes de usuários.
- Administração de Dados: quando diversos usuários compartilham dados, centralizar a administração dos dados pode oferecer melhorias significativas. Profissionais experientes que compreendem a natureza dos dados sendo gerenciados, e como os diferentes grupos de usuários os utilizam, podem ser responsáveis por organizar a representação dos dados para minimizar a redundância e para realizar as sintonizações finas do armazenamento dos dados para garantir uma eficiente recuperação.

- Acesso Concorrente e Recuperação de Falha: um SGBD planeja o acesso concorrente aos dados de maneira tal que os usuários podem achar que os dados estão sendo acessados por apenas um único usuário de cada vez. Além disso, o SGBD protege os usuários dos efeitos de falhas de sistema.
- Tempo Reduzido de Desenvolvimento de Aplicativo: obviamente, o SGBD suporta funções importantes que são comuns a vários aplicativos que acessam os dados no SGBD. Isso, em conjunto com uma interface de alto nível aos dados, facilita o desenvolvimento rápido de aplicativos. Os aplicativos de SGBD tendem a sermais robustos do que os aplicativos similares independentes porque muitas tarefas importantes são tratadas pelo SGBD (e não precisam ser depuradas e testadas no aplicativo).

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- Existe uma grande diferença entre dado, informação e conhecimento.
- Sempre tivemos a necessidade de armazenar as informações.
- Que os dados fazem parte do nosso cotidiano.
- Bancos de dados relacionais predominam no mundo do desenvolvimento de software.
- SGBD ou Sistema Gerenciador de Bancos de Dados foi uma evolução do paradigma de armazenamento em arquivos.

AUTOATIVIDADE

1 De acordo com o conteúdo estudado, defina o conceito de dado, informação e conhecimento.

2 Sistemas Gerenciadores de Banco de Dados (SGBD) são programas de computador que têm como principal objetivo facilitar a organização, o acesso e a manipulação dos dados, diminuindo, assim, a complexidade da aplicação referente à sua estrutura. Classifique V para as sentenças verdadeiras e F para as falsas:

( ) Podemos destacar do SGBD o backup, ferramenta para que o administrador do banco de dados possa proteger suas informações.
( ) Para a utilização SGBD, não é necessário utilizar autenticação, já que segurança não é uma das características desse software.
( ) De forma geral, os SGBD não possuem módulos de funcionalidades bem definidos.
( ) A organização de armazenamento dos dados de um SGBD é formada obrigatoriamente por apenas uma tabela de dados.

Agora, assinale a alternativa que apresenta a sequência CORRETA:

a) ( ) V - F - F - F.
b) ( ) V - F - V - F.
c) ( ) V - V - V - F.
d) ( ) F - V - F - V.

3 Para compreender melhor a respeito de banco de dados, é crucial atentarmos para alguns conceitos básicos. Entre esses conceitos, podemos destacar um item que tem como principal característica a de ser a menor unidade de conteúdo que, isoladamente, não tem nenhum significado. Com base nessa descrição, assinale a alternativa CORRETA:

a) ( ) Dado.
b) ( ) Conhecimento.
c) ( ) Informação.
d) ( ) Pesquisa.

4 Vários são os conceitos valiosos que ajudam na compreensão dos itens que compõem um banco de dados. Com relação a esses itens, há um conceito que se destaca pela característica de ser um conjunto de dados já processado e agora possui significado. A que se refere essa definição?

a) ( ) Informação.
b) ( ) Formulário de pesquisa.
c) ( ) Depósito de dados.
d) ( ) Pesquisa.

5 Segundo Elmasri e Navathe, um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de banco de dados entre vários usuários e aplicações. Acerca das principais funcionalidades do SGBD, analise as sentenças a seguir:

I- Manutenir o dicionário de dados, que é uma listagem organizada de todos os elementos de dados que são pertinentes ao sistema.
II- Controlar o acesso aos dados e a modificação dos bancos de dados aumentando a segurança e integridade destes.
III- Impedir o acesso simultâneo ao mesmo dado por meio de logs de gerenciamento.
IV- Definir a estrutura de armazenamento e o método de acesso aos dados.

Assinale a alternativa CORRETA:

FONTE: ELMASRI, Ramez; NAVATHE, Schmkant B. Sistemas de Banco de Dados. São Paulo: Editora Pearson, 2005.

a) ( ) As sentenças I e II estão corretas.
b) ( ) As sentenças I, II e III estão corretas.
c) ( ) Somente a sentença II está correta.
d) ( ) As sentenças II, III e IV estão corretas.

TÓPICO 2

MODELOS DE REPRESENTAÇÃO EM BANCOS DE DADOS

1 INTRODUÇÃO

O grande objetivo de um sistema de banco de dados é prover os usuários com uma visão abstrata dos dados. Isto é, o sistema omite certos detalhes de como os dados são armazenados e mantidos. Entretanto, para que o sistema possa ser utilizado, os dados devem ser buscados de forma eficiente. Este conceito tem direcionado o projeto de estrutura de dados complexas para a representação de dados em um banco de dados. Uma vez que muitos dos usuários de banco de dados não são treinados para computação, a complexidade está escondida deles através de diversos níveis de abstração que simplificam a interação do usuário com o sistema. Antes de começar a modelagem dos dados relacional, vamos conhecer um pouco das regras que dão base para o banco.

Inicialmente, vamos conhecer um pouco sobre Codd, segundo Souza (2015), ele nasceu na Inglaterra em 19 de agosto de 1923, filho de uma professora e de um artesão coureiro. Ele lutou na Segunda Guerra Mundial. Mas foi nos Estados Unidos que ele exerceu sua principal atividade intelectual e profissional, maior parte do tempo, como funcionário da IBM. Empresa essa que não quis aplicar as suas ideias. Ele, porém, procurou alguns clientes da empresa para apresentar as vantagens de sua proposta, o que forçou a IBM a lançar a tecnologia oficialmente. A ideia original, porém, se desvirtuou devido ao fato dos desenvolvedores da IBM não a compreenderem bem. Com isso, acabou surgindo o SQL. Nos anos de 1981, Codd recebia o Prêmio Turing, onze anos após a publicação do artigo em que apresentou o modelo relacional. O conceito que foi criado por Edgar Frank Codd em 1970 e descrito no artigo "*Relational Model of Data for Large Shared Data Banks*". Foi o primeiro modelo de dados descrito teoricamente. Descontente com a IBM, ele sairia da empresa junto com um colega, ele montaria uma nova empresa Codd & Date.

O modelo relacional que Codd propôs, segundo o site Devmedia (2011), representa um modelo de dados utilizado em Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDRS). Segundo o próprio Codd referenciado por Souza em sua dissertação de mestrado, que estuda a história de Codd, afirma que:

> A motivação mais importante para o trabalho de investigação que resultou no modelo relacional foi o objetivo do fornecer uma fronteira nítida e clara entre a lógica e aspectos físicos de gestão de banco de dados (incluindo projeto de banco de dados, recuperação de dados e manipulação de dados) (CODD, 1981 *apud* SOUZA, 2015, p. 25).

Conheça um pouco mais sobre a história de Codd em: https://tede2.pucsp.br/handle/handle/13305.

Vale ressaltar que, segundo Costa (2012, s.p.), as doze regras de Codd estão baseadas na regra zero, que determina que, "Qualquer sistema considerado, ou que deseja ser um sistema gerenciador de banco de dados relacional deve ser capaz de gerenciar, por completo, bases de dados através de sua capacidade relacional". Em suma, essa regra determina que um SGBDR não permite exceções quanto ao modelo relacional de gerenciamento de bases de dados. Vamos as regras:

2 AS DOZE REGRAS DE CODD

Vamos conhecer as regras propostos por Codd, segundo a leitura complementar no Livro Didático de Princípios de Banco de Dados:

Aprofunde seus conhecimentos e leia esse livro também: LEHMKUHL, Décio; EGER, Djayson Roberto.**Princípios de banco de dados**. Indaial: Uniasselvi, 2013.

- Regra 1: regra das informações em tabelas: As informações a serem armazenadas no banco de dados devem ser apresentadas como relações (tabelas formadas por linhas e colunas) e o vínculo de dados entre as tabelas deve ser estabelecido por meio de valores de campos comuns (chaves estrangeiras).

- Regra 2: regra de acesso garantido: Todo e qualquer valor atômico em um BD relacional possui a garantia de ser logicamente acessado pela combinação do nome da tabela, do valor da chave primária e do nome do campo/coluna que deseja acessar. Isso porque, com o nome da tabela, se localiza a tabela desejada. Com o valor da chave primária a tupla desejada dentro da tabela é localizada. E com o nome do campo/coluna se acessa a parte desejada da tupla.

- Regra 3: regra de tratamento sistemático de valores nulos: Valores nulos devem ser suportados de forma sistemática e independente do tipo de dado para representar informações inexistentes e informações inaplicáveis. Deve-se sempre lembrar que valores nulos devem ter um tratamento diferente de "valores em branco".

- Regra 4: Regra do catálogo relacional ativo: Toda a estrutura do banco de dados (domínios, campos, tabelas, regras de integridade, índices etc.) deve estar disponível em tabelas (também referenciadas como catálogo). Sua manipulação é possível por meio de linguagens específicas (por exemplo, SQL). Essas tabelas são, geralmente, manipuladas pelo próprio sistema no momento em que o usuário efetua alterações na estrutura do banco de dados (por exemplo, a inclusão de um novo atributo em uma tabela).

- Regra 5: regras de atualização de alto nível: Essa regra diz que o usuário deve ter capacidade de manipular as informações do banco de dados em grupos de registros, ou seja, ser capaz de inserir, alterar e excluir vários registros ao mesmo tempo.

- Regra 6: regra de sublinguagem de dados abrangente: Pelo menos uma linguagem, com sintaxe bem definida, deve ser suportada, para que o usuário possa manipular a estrutura do banco de dados (como criação e alteração de tabelas), assim como extrair, inserir, atualizar ou excluir dados, definir restrições de integridade e de acesso e controle de transações (commit e rollback, por exemplo). Deve ser possível ainda a manipulação dos dados por meio de programas aplicativos.

- Regra 7: regra de independência física: Quando for necessária alguma modificação na forma como os dados estão armazenados fisicamente, nenhuma alteração deve ser necessária nas aplicações que fazem uso do banco de dados (isolamento), assim como devem permanecer inalterados os mecanismos de consulta e manipulação de dados utilizados pelos usuários finais.

- Regra 8: regra de independência lógica: Qualquer alteração efetuada na estrutura do banco de dados como inclusão ou exclusão de campos de uma tabela ou alteração no relacionamento entre tabelas não deve afetar o aplicativo utilizado ou ter um baixo impacto sobre o mesmo. Da mesma forma, o aplicativo somente deve manipular visões dessas tabelas.

- Regra 9: regra de atualização de visões: Uma vez que as visões dos dados de uma ou mais tabelas são, teoricamente, suscetíveis a atualizações, então um aplicativo que faz uso desses dados deve ser capaz de efetuar alterações, exclusões e inclusões neles. Essas atualizações, no entanto, devem ser repassadas automaticamente às tabelas originais. Ou seja, a atualização em uma visão deve refletir na atualização das tabelas representadas por essa visão.
- Regra 10: regra de independência de integridade: As várias formas de integridade de banco de dados (integridade de entidade, integridade referencial e restrições de integridade complementares) precisam ser estabelecidas dentro do catálogo do sistema ou dicionário de dados e serem totalmente independentes da lógica dos aplicativos. Assim, os aplicativos não devem ser afetados quando ocorrerem mudanças nas regras de restrições de integridade.

- Regra 11: regra de independência de distribuição: Alguns SGBDs, notadamente os que seguem o padrão SQL, podem ser distribuídos em diversas plataformas/ equipamentos que se encontrem interligados em rede. Essa capacidade de distribuição não pode afetar a funcionalidade do sistema e dos aplicativos que fazem uso do banco de dados. Em resumo, as aplicações não são logicamente afetadas quando ocorrem mudanças geográficas dos dados (caso dos BDs distribuídos).

- Regra 12: regra não subversiva: O sistema deve ser capaz de impedir qualquer usuário ou programador de transgredir os mecanismos de segurança, regras de integridade do banco de dados e restrições, utilizando algum recurso de linguagem de baixo nível que eventualmente possam ser oferecidos pelo próprio sistema.

Sua estrutura classifica dados em tabelas, também conhecidas como relações, cada uma das quais consiste em colunas e linhas. Cada coluna lista um atributo da entidade em questão. Em cada linha, também chamada de tupla, inclui dados sobre uma instância específica da entidade em questão, como um determinado colaborador. Este modelo foi formulado por Codd, tendo sua teoria sustentada pela álgebra relacional.

Na Unidade 2 do nosso caderno, vamos conhecer a álgebra relacional.

Tendo o conhecimento dos alicerces dos bancos de dados relacionais, vamos começar a etapa de modelagem dos dados, para que ela seja compreensível por todos os interessados no projeto.

Como vimos, nem sempre os nossos clientes ou usuários, tem conhecimento sobre tecnologia, por isso temos que ter em mente, que para obtermos sucesso no desenvolvimento de algum projeto que tenha interação com o banco e dados, é de suma importância que haja o conhecimento sobre à modelagem de dados. Uma modelagem bem realizada, irá garantir que os dados que serão utilizados estão consistentes, estão armazenados da melhor maneira com o objetivo de ter um bom desempeno além da integridade das informações. Ele será um espelho de como as informações serão armazenadas no futuro banco de dados. Quanto melhor e mais correta estiver a modelagem, as chances de sucesso no projeto de banco de dados aumentam.

Caso essa modelagem não seja feita de forma fiel ao ambiente em análise, ou seja, nosso minimundo, o sucesso do projeto pode estar em risco. Pois podem ocorrer conflitos de integridade, baixo desempenho, além da duplicidade de informações.

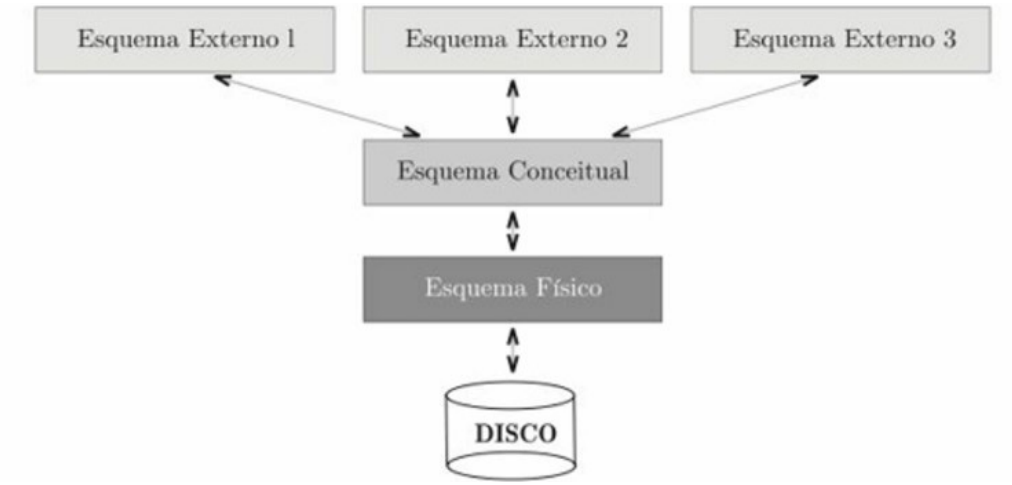
Por isso, o principal objetivo da modelagem é gerar uma abstração da realidade que seja capaz de registrar os acontecimentos, de forma que seja possível desenvolver um sistema que resolva as necessidades e mantenha as informações esperadas e necessárias.

Para facilitar a comunicação e o entendimento, durante a modelagens, não se usa códigos ou termos técnicos, ela utiliza notações em idioma, porisso, é recomendado que o modelo de dados, depois de elaborado, sejam revisados e verificados procurando anomalias ou possíveis problemas de entendimento entre a equipe de desenvolvimento e o cliente.

Essa representação dos dados pode estar submetida a diferentes níveis de abstração, sendo eles classificados como: modelos conceituais, modelos lógicos e modelos físicos. Veremos mais detalhes durante esta unidade.

Não iremos detalhar os modelos de dados, pois pretendemos dar um embasamento para que você tenha as condições e conhecimentos mínimos para projetar um banco de dados relacional dentro da metodologia e segurança necessária para o sucesso de uma aplicação.

FIGURA 9 – NÍVEIS DE ABSTRAÇÃO EM UM SGBD



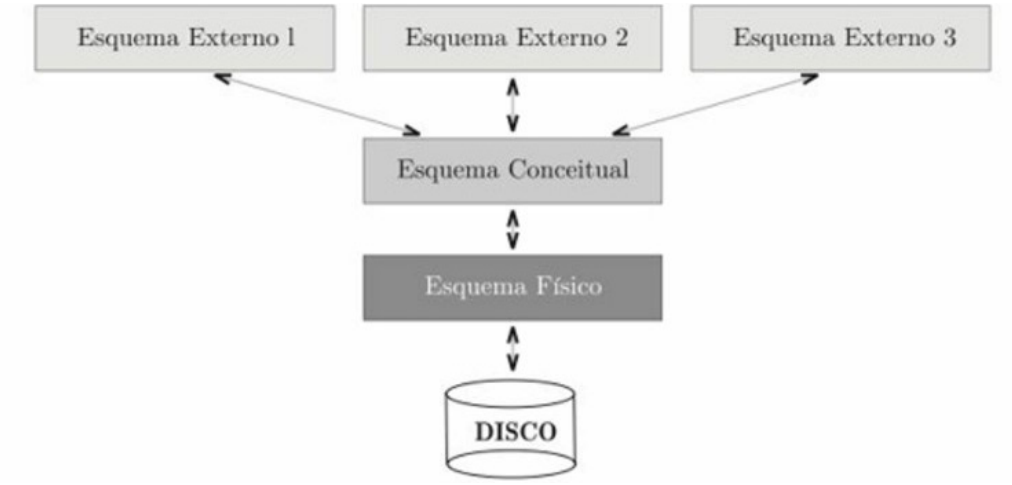
FONTE: Ramakrishnan (2011, p. 10)

- MODELO CONCEITUAL

Como já mencionamos, o modelo conceitual é uma descrição mais abstrata da realidade, onde os fatos do mundo real são apresentados de uma forma mais natural, bem como suas propriedades e relacionamentos. Ele é utilizado para entendimento, transmissão, validação de conceitos e mapeamento do ambiente, possibilitando um melhor diálogo entre todos os envolvidos.

Devido a sua característica esclarecedora, o modelo conceitual é uma definição em alto nível, que irá retratar o contexto, o processo de negócio que será implementado. Por isso, ele deve ser o primeiro passo a ser desenvolvido em um projeto de banco de dados. Neste momento é feita de forma simples e de fácil entendimento o contexto de negócio, para futura implementação em banco.

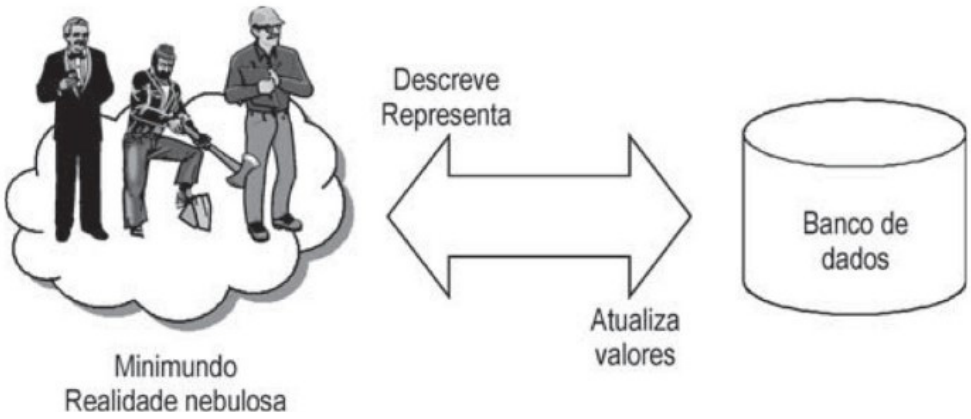
FIGURA 10 – NÍVEIS DE ABSTRAÇÃO EM UM SGBD



FONTE: Adaptado de Ramakrishnan (2011, p. 10)

A primeira coisa que precisamos conhecer antes de entrar nos níveis de abstração, temos que delimitar qual será o minimundo. Segundo Machado (2014) é ele que vai dar o tamanho do nosso projeto, pois ele é uma porção da realidade, que é captada pelo analista, que a gestão de negócios de uma organização tem interesse em observar, controlar e administrar. A complexidade existente no momento de analisar um minimundo pode levar o analista a subdividi-lo em partes menores, que damos o nome de visão de processo de negócio. Ele representa algum aspecto do mundo real, que é chamado de minimundo, qualquer alteração efetuada no minimundo é automaticamente refletida no banco de dados.

FIGURA 11 – DELIMITAÇÃO DO MINIMUNDO



FONTE: Machado (2014, p. 19)

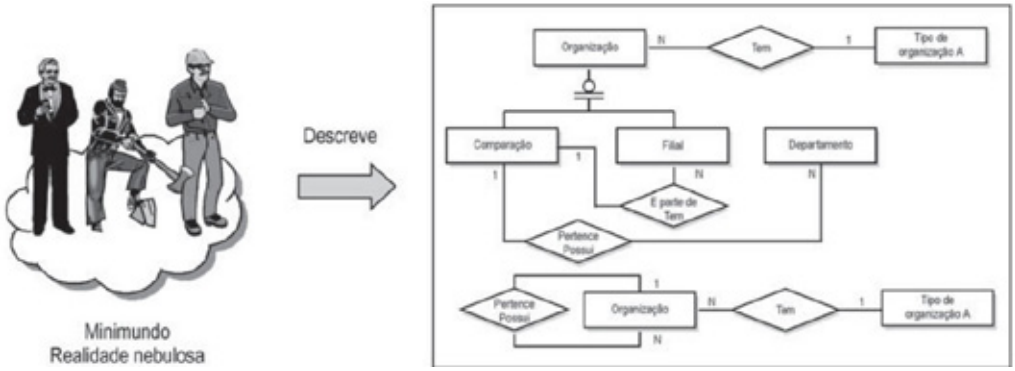
E através desse recorte que definiremos nosso modelo de dados, para conseguir explicar –para ilustrar e melhor comunicar-se com o usuário o minimundo, as limitações, por exemplo, modelar o departamento de finanças, ou compras. Ou seja, esse modelo tem como objetivo descrever de forma simples e facilmente compreendida pelo usuário final as informações de um contexto de negócios, as quais devem ser armazenadas em um banco de dados.

Segundo Machado (2014), a definição de um modelo conceitual é uma descrição de alto nível, uma macrodefinição, que tem a preocupação de captar e retratar toda a realidade que será modelada. Vale destacar que esse modelo não retrata nem é vinculado aos aspectos ligados à abordagem do banco de dados que será utilizado e também não se preocupa com as formas de acesso ou estruturas físicas implementadas por um SGBD específico. **Sem modelo conceitual não temos uma visão clara das regras do negócio e acabamos por criar aplicações sem entender para que elas foram criadas**.

O resultado desse modelo, continuando com Machado (2014), é um esquema gráfico que representa a realidade das informações existentes em determinado contexto de negócios, assim como as estruturas de dados em que estão organizadas essas informações. Vale ressaltar, que esse modelo nunca deve ser construído com considerações sobre processos de negócio, com preocupações de

acesso aos dados, não devendo existir preocupação alguma com o modo como serão realizadas as operações de consulta e manutenção dos dados nele apresentados. O seu foco deve ser direcionado sempre ao entendimento e à representação de uma realidade, de um contexto. O seu resultado é um esquema gráfico que representa a realidade das informações existentes no minimundo contextualizado, ou seja, o contexto de negócios, assim como as estruturas de dados em que estão organizadas essas informações. Ele também não deve ser desenvolvido preocupando como será o acesso aos dados, nem como serão realizadas as operações e manutenções. O foco deve ser orientado sempre ao entendimento e à representação de uma realidade, de um contexto.

FIGURA 12 – MODELO CONCEITUAL



FONTE: Machado (2014, p. 19)

Durante o processo de concepção do modelo conceitual, o analista deverá analisar fatos e buscar documentações, registros em outros sistemas que estarão se relacionando, regras de negócios, tudo que estiver relacionado e que possa lhe auxiliar em um melhor entendimento do processo que está sendo desenvolvido. Segundo Lehmkuhl (2013), é importante que o processo foque em fatos relevantes e que serão utilizados para a geração de informações e registro no sistema proposto. Lembrando que o Modelo Conceitual não está relacionado com o modelo de banco de dados, forma de acesso ou armazenamento dos dados. Ele faz uma representação gráfica de uma realidade existente em um contexto de negócio, ou no minimundo analisado, conforme apresentado na figura acima.

- MODELO LÓGICO

É a etapa intermediaria entre o usuário e a efetivação dos dados no banco. O modelo lógico somente será desenvolvido após a criação do modelo conceitual, segundo Machado (2014), pois neste momento começamos a analisar uma das abordagens possíveis da tecnologia Sistemas Gerenciadores de Banco de Dados (relacional, hierárquica, rede ou orientada a objetos) para estruturação e estabelecimento da lógica dos relacionamentos existentes entre os dados definidos no modelo conceitual.

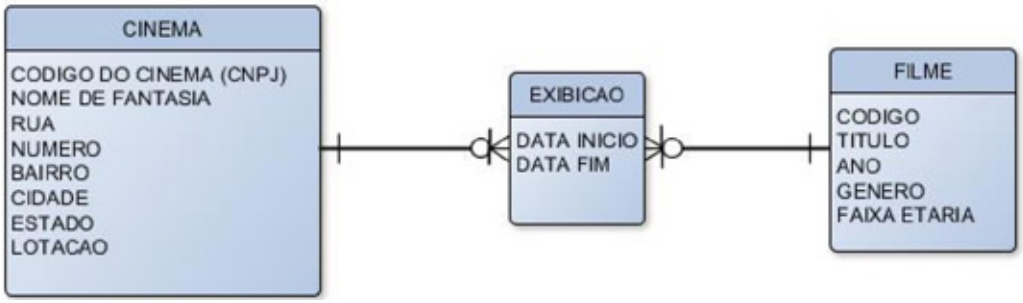
Essa sequência de modelos, começando pelo conceitual e seguido pelo lógico, se faz necessária, pois caso fôssemos direto para a construção do modelo lógico à vinculação tecnológica de nosso raciocínio, acabaria perturbando a interpretação pura e simples de um contexto, mesmo se já sabemos a abordagem para banco de dados, para que estamos realizando um projeto.

Machado (2014) alerta para que sempre que analisamos um contexto sob a ótica tecnológica, tendemos a sermos técnicos demais, desviar a realidade, conduzindo-a às limitações da tecnologia empregada. E isso já foi comprovado, que induz a erros de interpretação da realidade, criando assim modelos de dados que não possuem aderência ao minimundo descrito.

O modelo lógico, de acordo com Machado (2014), apresenta em formato as estruturas que estarão no banco de dados de acordo com as possibilidades permitidas pela sua abordagem, sem ainda considerar nenhuma característica específica de um SGBD. O que resulta em um esquema lógico de dados sob a visão de uma das abordagens citadas, pelo emprego de uma técnica de modelagem de dados orientada às restrições de cada abordagem.

Semelhante a visão apresentada, Lehmkuhl (2013) afirma que o modelo lógico objetiva a representação das estruturas que irão armazenar os dados. Neste momento ocorre a definição das entidades e dos atributos. O objetivo do modelo de dados lógico é a representação dos dados em uma estrutura de armazenamento de dados. Nesta etapa é definida a estrutura de registro do Banco de Dados conforme a figura a seguir:

FIGURA 13 – MODELO LÓGICO



FONTE: Lehmkuhl (2013, p. 75)

A próxima etapa em um modelo de dados relacional é a efetiva transformação em modelo do modelo lógico para o modelo físico.

- MODELO FÍSICO

Após ter desenvolvido o modelo conceitual e posteriormente o modelo lógico, o modelo agora já pode ser transformado de uma entidade para uma tabela efetivamente no banco de dados. Nesta etapa já se tem um modelo lógico

definido, com intuito de ser aplicado sobre um SGDB. Neste momento entram as questões relacionadas ao tipo e tamanho do campo, relacionamento, indexação, restrições etc. Ele descreve as estruturas físicas de armazenamento, tais como tabelas, índices, gatilhos, funções, visões, nomenclaturas etc.

Machado (2014) descreve que o modelo físico é projetado de acordo com os requisitos de processamento e uso mais econômico dos recursos computacionais. Esse modelo detalha o estudo dos métodos de acesso do SGBD para criação dos índices necessários para cada informação colocada nos modelos conceitual e lógico. Em ambiente de banco de dados relacional denominamos script de criação do banco de dados o conjunto de comandos em SQL (DDL), que será executado no Sistema Gerenciador de Banco de Dados para a criação de banco de dados correspondente ao modelo físico.

Essa é a etapa final do projeto de banco de dados, em que será utilizada a linguagem de definição de dados do SGDB, a DDL, para a construção do banco de dados com base no script SQL gerado. Neste caso, o script pode variar um pouco, dependendo de qual banco será utilizado.

Neste livro didático utilizaremos o banco de dados Oracle. Faremos uma breve apresentação dos tipos de dados, para o entendimento do script, pois eles serão detalhados mais adiante.

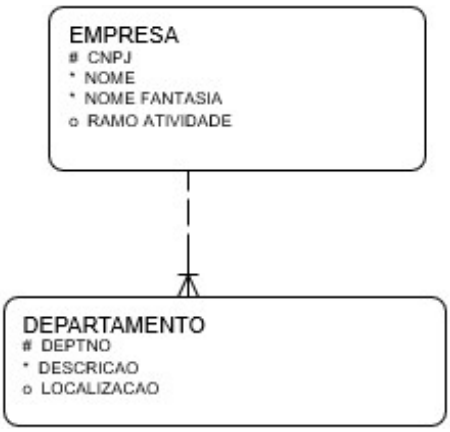
QUADRO 1 – TIPOS DE DADOS

Tipos de dados	Descrição
`NUMBER(p,s)`	Dados numéricos de comprimento variável
`VARCHAR2(tamanho)`	Dados de caractere de comprimento variável
`CHAR (tamanho)`	Dados de caractere de comprimento fixo
`DATE`	Valores de data e hora

FONTE: O autor

O modelo lógico que será transformado em físico, conforme a figura a seguir:

FIGURA 14 – EXEMPLO MODELO LÓGICO



FONTE: O autor

A seguir temos o modelo de um script que transforma o modelo apresentado na figura anterior em um modelo físico, ou seja, já é possível de ser executado e criado no banco de dados.

FIGURA 15 – MODELO FÍSICO

```
CREATE TABLE departamento
 (deptno   NUMBER(3) CONSTRAINT dept_deptno_pk PRIMARY KEY,
 descricao VARCHAR2(14),
 localizacao VARCHAR2(13));

CREATE TABLE empresa(
 cnpj NUMBER(10) ONSTRAINT emp_ cnpj_pk PRIMARY KEY,
 deptno NUMBER(3) NOT NULL);
 nome VARCHAR2(40) not null,
 nome_fantasia VARCHAR2(40) not null,
 ramo_atividade VARCHAR2(20));
```

FONTE: O autor

A Unidade 3 será toda dedicado à prática em criamos tabelas e trabalharmos com os dados.

Agora já sabemos quais são os modelos, vamos voltar e aprender o que é necessário para a criação de um modelo de entidade e relacionamento.

Antes de tudo, vamos definir o que é um MER. É o conjunto de conceitos e elementos de modelagem que o projetista de banco de dados precisa conhecer, já o DER, será o resultado do processo de modelagem executado pelo projetista de dados que conhece o MER.

2.1 MER (MODELO ENTIDADE RELACIONAMENTO) E DER (DIAGRAMA ENTIDADE-RELACIONAMENTO)

Antes de efetivamente criarmos as tabelas no banco, devemos inicialmente entender o contexto da aplicação, ou o nosso minimundo a ser modelado. Um dos primeiros passos a ser realizado é o estudo e levantamento dos requisitos necessários. Durante essa etapa, identificam-se as principais partes e objetos envolvidos, suas possíveis ações e responsabilidades, suas características e como elas interagem entre si.

Feito isso, partimos para o desenvolvimento do modelo conceitual que será utilizado para orientar o desenvolvimento, gerando as informações sobre os aspectos relacionados ao domínio do projeto em questão.

2.1.1 Modelo Entidade Relacionamento (MER)

Conhecido também como Modelo ER, ou simplesmente MER, ele é um modelo conceitual utilizado para descrever os objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como elas se relacionam entre si (relacionamentos).

Normalmente, este modelo representa de forma abstrata a estrutura que será usada banco de dados da aplicação. Certamente, o banco não serão apenas estas entidades que o banco terá, ele poderá ter outras entidades, tais como chaves e tabelas intermediárias, que podem só fazer sentido no contexto de bases de dados relacionais. Vamos começar entendendo cada uma das partes desse modelo.

Esse modelo fará parte da documentação e das plantas que são base para os engenheiros de software que estão desenvolvendo o projeto ou qualquer outro membro da equipe ou cliente. Ele servirá para que se consiga construir o sistema de forma funcional e segura, garantindo as funcionalidades solicitadas pelo cliente sejam atendidas, uma vez que todos têm o mesmo entendimento. O MER deve apresentar os seguintes elementos básicos:

- Entidades.
- Atributos.
- Relacionamentos.

Considere que as entidades são os objetos, cujo envolvimento em um domínio de negócios deve ser descrito, já os atributos são as características de cada objeto ou entidade presente no modelo e os relacionamentos serão as conexões que esses objetos ou entidades realizam entre si. Vamos detalhar cada um deles.

- Entidades: segundo o site Devmedia (2014), os objetos ou partes envolvidas um domínio, também chamados de entidades, podem ser classificados como físicos ou lógicos, de acordo sua existência no mundo real. Reforçando, as entidades correspondem aos objetos envolvidos em um ambiente. Esses objetos irão realizar conexões com os outros objetos para trocar informações necessárias para o correto funcionamento do sistema que será construído. As entidades se subdividem em: entidades físicas e entidades lógicas.
- Entidades físicas: as entidades, segundo Devmedia (2014) que são classificadas como físicas, são aquelas realmente tangíveis e que existem no mundo real, como exemplo: clientes (uma pessoa, uma empresa), produtos (um móvel, um celular, uma roupa) etc. Ao falarmos de entidades físicas, são coisas que têm existência no mundo real.
- Entidades Lógicas: diferente da entidade física que são coisas do mundo real, as entidades lógicas são aquelas que existem geralmente como resultado da interação entre ou com entidades físicas, que fazem sentido dentro de um certo domínio de negócios, mas que no mundo externo/real não são objetos físicos. Vejamos a definição Segundo Barboza (2018, p. 62):

> São exemplos disso uma venda ou uma classificação de um objeto (modelo, espécie, função de um usuário do sistema). Essa necessidade se dá em virtude dos relacionamentos ou da classificação sobre as outras entidades ou conexões entre elas. Exemplos de entidades lógicas podem ser grupos de usuários do sistema, vendas, relatório etc. A entidade grupos de usuários do sistema existe para regular as permissões dos usuários ao sistema; a entidade venda é necessária para relacionar a conexão de entrega de algo da entidade física, produto, para alguma pessoa ou empresa da entidade física cliente, guardando os dados dessa entrega, como valores, data etc.; por fim, a entidade relatório existe para puxar informações da própria entidade lógica venda e apresentar os dados ao usuário ou gerente que a solicitou.

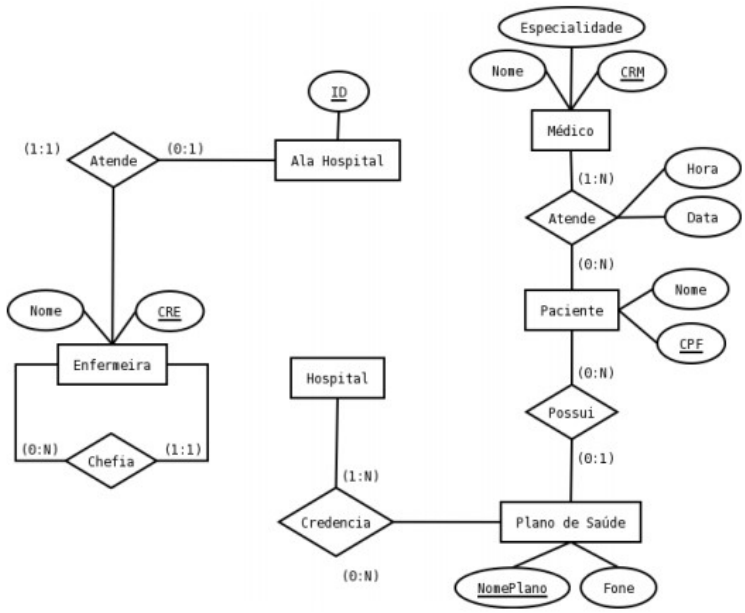
Agora que já conhecemos os dois tipos de entidades, vamos conhecer a sua classificação na próxima seção, até lá vamos praticar os conhecimentos aprendidos. Vamos ver um exemplo, proposto por Rocha (2012):

Considere o banco de dados de um hospital. De acordo com os requisitos a seguir, utilize o MER pararepresentar o banco de dados.

- O hospital possui várias alas.
- Cada ala possui uma enfermeira responsável.
- Cada enfermeira se reporta a uma enfermeira-chefe.
- Cada ala possui uma enfermeira responsável.
- Cada enfermeira se reporta a uma enfermeira-chefe.
- Enfermeiras podem atender apenas uma ala.
- O hospital atende (credencia) os planos de saúde A, B e C.
- Para cada plano de saúde, énecessário saber os m´médicos credenciados no mesmo.
- Médico tem CRM e enfermeira CRE que lhes são únicos.
- Todo atendimento de um médico a um paciente deve ser registrado com a data e hora em que omesmo ocorreu.
- Um mesmo paciente pode ser atendido por mais de um médico.
- Hospital tem CNPJ.
- Ala do hospital tem um identificador.
- Plano de saúde tem um nome e telefone da operadora.
- Médicos têm nome e especialidade.
- Enfermeiras têm nome.
- O nome de um plano de saúde é único.

Vejamos a solução proposta:

FIGURA 16 - MODELO ER HOSPITAL



FONTE: Rocha (2012, s.p.)

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- Para um banco ser considerado relacional, deverá atender aas regras de Codd.
- A modelagem de dados se divide em três etapas: Modelo Conceitual, Modelo Lógico, Modelo Físico.
- Enquanto o MER é um modelo conceitual, o Diagrama Entidade Relacionamento (Diagrama ER ou ainda DER) é a sua representação gráfica e principal ferramenta.

AUTOATIVIDADE

1 Em relação ao MER, podemos afirmar que:

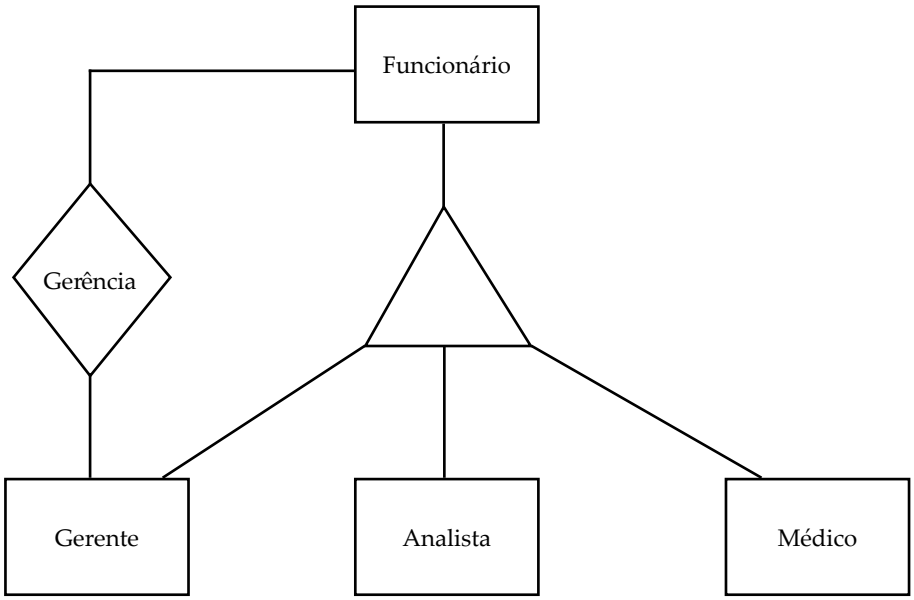
I- Baseia-se em uma percepção do mundo real, retratando uma coleção de conceitos ou objetos com características e relacionamentos entre si.
II- Retrata um conjunto de entidades que possuem características denominadas, métodos as quais interferem no seu modo de comportamento, frente a alterações e interações com outras entidades do banco.
III- Pode ser expresso graficamente através do diagrama E-R, tendo como componentes retângulos, elipses, losangos e linhas.

É correto o que se afirma em:
a) ( ) I e II.
b) ( ) I, apenas.
c) ( ) II, apenas.
d) ( ) I e III.
e) ( ) II e III.

2 Foi solicitada a criação de um Modelo de entidade relacionamento (MER) para uma livraria. De acordo com os requisitos a seguir:

- Deve-se manter um cadastro de clientes, que pode ser física (CPF) ou jurídica (CNPJ).
- Cada cliente, só pode ter um único cadastro.
- Para cada cliente, deve-se armazenar seu endereço, telefone, CPF e lista dos livros que comprou. E para cada compra, registar a data em que esta foi realizada.
- Sabe-se que um cliente pode comprar muitos livros, e que um livro pode ser vendido para mais de um cliente, uma vez que a livraria mantém um estoque.
- A livraria compra livros diretamente de editoras. E Editoras diferentes não fornecem o mesmo tipo de livro.
- Sobre as editoras, a livraria precisa de seu código, endereço, telefone de contato, e o nome de seugerente.
- Deve-se manter um cadastro sobre cada livro na livraria. Para cada livro, é importante armazenaro nome do autor, assunto, editora, ISBN e a quantidade dos livros em estoque.

3 De acordo com o DER a seguir, um analista ou médico não podem ser gerentes. Por quê? Qual é a alteração necessária para tornar isso possível?



TÓPICO 3

ENTIDADES, ATRIBUTOS E RELACIONAMENTOS

1 INTRODUÇÃO

Quando começamos o levanto das informações para a criação de um novo sistema, ou algum incremento em algum sistema já existente, uma das primeiras atividades a ser desenvolvida é o estudo e o levantamento dos requisitos relacionados ao projeto.

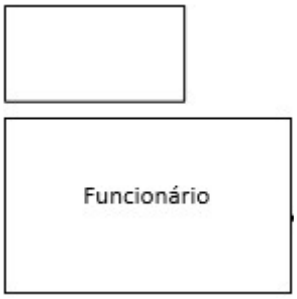
Com base nessas informações obtidas, consegue se desenvolver um modelo conceitual que será empregado na orientação e no desenvolvimento, fornecendo informações sobre as características do projeto, através da identificação das entidades, sejam elas novas ou as antigas**.**

Após a identificação, faremos a classificação dessas entidades, dividimos elas em entidades fortes, entidades fracas e entidades associativas. Além dessa classificação das entidades, classificamos os relacionamentos dos atributos e dos relacionamentos. Vamos começar classificando as entidades.

2 ENTIDADES FORTES

Entidades fortes, de acordo com Devmedia (2014) são as que justificam sua existência no MER e não dependem de outras, elas já têm total razão em existir, devido a sua importância e não dependem de outros fatores ou entidades. Vamos exemplificar, imagine que esteja modelando um sistema de armazenamento e vendas, a entidade produto, por exemplo, independe de quaisquer outras para existir. Na Figura 18, temos a representação gráfica de uma entidade forte, que é um retângulo. Temos também outro exemplo, pois a entidade funcionário, pode existir sozinha, e não depende de outra.

FIGURA 17 – REPRESENTAÇÃO DE ENTIDADE FORTE

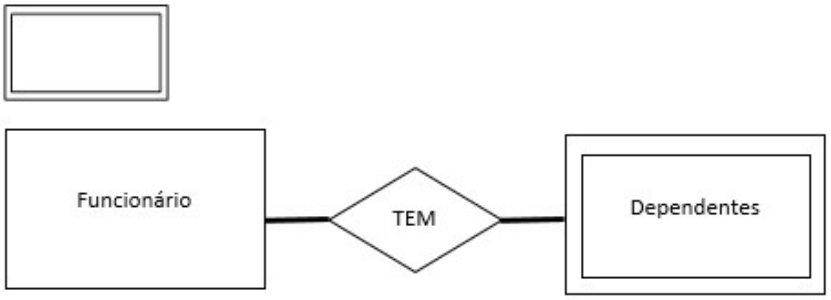


FONTE: O autor

3 ENTIDADES FRACAS

Ao contrário das entidades fortes, as entidades fracas são aquelas que dependem de outras entidades para existirem, pois individualmente elas não fazem sentido. Mantendo o mesmo exemplo, a entidade venda depende da entidade produto, pois uma venda sem itens não tem sentido. Que é representada por um retângulo dentro do outro, e um exemplo da entidade dependente, que tem sua existência atrelada à entidade funcionário. Não teria sentido sua existência isolada.

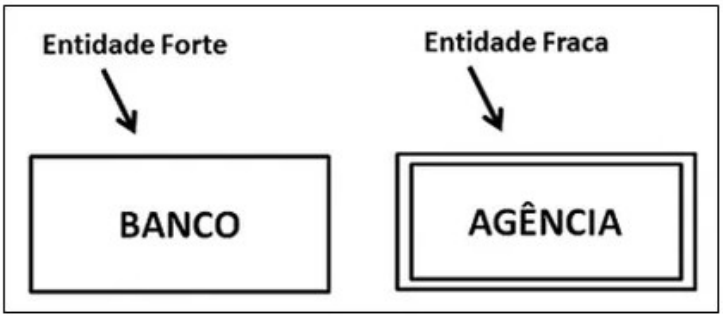
FIGURA 18 – REPRESENTAÇÃO DE ENTIDADE FRACA



FONTE: O autor

Vejamos exemplos de entidades forte e fraca. Como já mencionamos a entidade fraca depende da forte para sua existência.

FIGURA 19 – ENTIDADE FORTE E FRACA



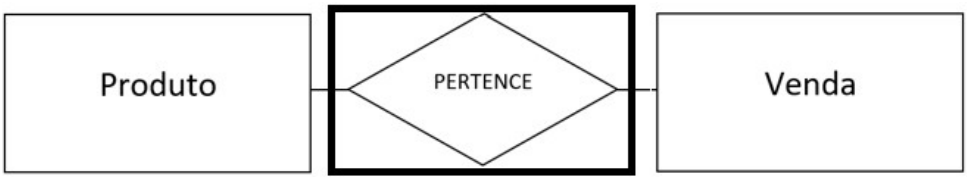
FONTE: O autor

O próximo modelo de entidade é a entidade associativa, que semelhante à entidade fraca, ela também é resultado de outra entidade.

4 ENTIDADES ASSOCIATIVAS

A entidade associativa, quando há necessidade de associar uma entidade a um relacionamento existente, ou seja, elas só existem se alguma associação ao relacionamento for necessária. Isso acontece porque o MER não admite que uma entidade se associe a um relacionamento. Por exemplo, eu tenho uma entidade que armazena os produtos, e outra que irá controlar a venda. Mas eu vou precisar de outra entidade, em que eu consiga identificar, em quais vendas determinados produtos foram vendidos.

FIGURA 20 – ENTIDADE ASSOCIATIVA



FONTE: O autor

Até este momento conhecemos as entidades e vimos que há linhas que as unem e chamamos de relacionamentos, que é o nosso próximo tema.

5 RELACIONAMENTOS

Como o próprio nome já diz, neste tópico veremos como identificar, analisar e compreender os relacionamentos que ocorrem entre as diversas entidades. Os Relacionamentos são associações entre entidades e possuem um significado específico dentro do mundo real.

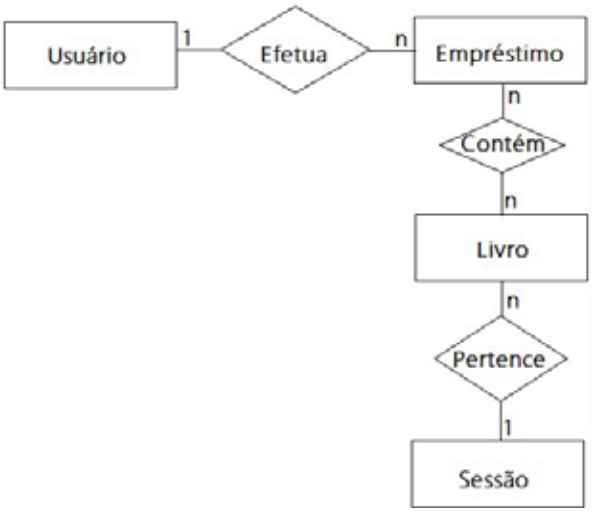
Para Heuser (2009), uma das propriedades sobre as quais pode ser desejável manter informações é a associação entre objetos. Exemplificando, pode ser desejável saber a que departamentos as pessoas estão associadas em uma organização. A propriedade de entidade que especifica as associações entre objetos é o relacionamento. A figura a seguir apresenta duas entidades, PESSOA e DEPARTAMENTO, e um relacionamento, LOTAÇÃO. Este modelo expressa que o BD mantém informações sobre um conjunto de objetos classificados como pessoas (entidade PESSOA), um conjunto de objetos classificados como departamentos (entidade DEPARTAMENTO) e um conjunto de associações, cada uma ligando um departamento a uma pessoa (relacionamento LOTAÇÃO).

FIGURA 21 – EXEMPLO RELACIONAMENTO

FONTE: Heuser (2009, p. 36)

Vale lembrar que os relacionamentos em geral são nomeados com verbos ou expressões que representam a forma como as entidades interagem, ou a ação que uma exerce sobre a outra. Essa nomenclatura pode variar de acordo com a direção em que se lê o relacionamento. No exemplo a seguir, temos um modelo de uma biblioteca, onde um usuário efetua o empréstimo de um ou vários livros. Um empréstimo pode conter vários livros, e vários livros podem pertencer a uma sessão, conforme apresentado na figura a seguir:

FIGURA 22 – EXEMPLO CARDINALIDADE



FONTE: <https://bit.ly/2UKqzMe>. Acesso em: 3 ago. 2019.

Na notação proposta por Peter Chen, as entidades deveriam ser representadas por retângulos, seus atributos por elipses e os relacionamentos por losangos, ligados às entidades por linhas, contendo também sua cardinalidade. Porém, notações mais modernas abandonaram o uso de elipses para atributos e passaram a utilizar o formato mais utilizado na UML, em que os atributos já aparecem listados na própria entidade.

Como vimos no exemplo, após feita a identificação das entidades são identificadas, deve-se então decidir como se dará o relacionamento entre elas. De acordo com a quantidade de objetos envolvidos em cada lado do relacionamento, podemos classificá-los em três formas (1..1, 1..n ou n..n) que são conhecidos por relacionamento binário e veremos o relacionamento ternário, que detalharemos a seguir:

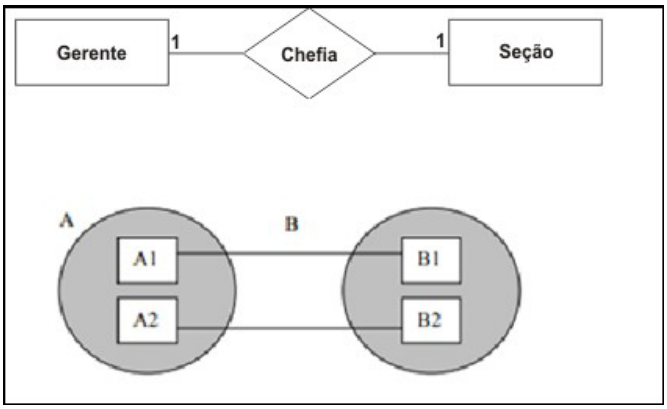
6 CARDINALIDADE MÁXIMA E MÍNIMA

Segundo Heuser (2009), para fins de projeto de banco de dados, uma propriedade importante de um relacionamento é a de quantas ocorrências de uma entidade podem estar associadas a uma determinada ocorrência através do relacionamento. Esta propriedade é chamada de cardinalidade de uma entidade em um relacionamento. Há duas cardinalidades a considerar: a cardinalidade máxima e a cardinalidade mínima.

Relacionamento 1..1 (um para um): no relacionamento 1..1 (um para um), cada uma das duas entidades envolvidas referencia obrigatoriamente apenas uma unidade da outra uma entidade, ou seja, a entidade A está associada no máximo a uma entidade B e uma entidade B está associada no máximo à entidade de A.

Por exemplo, um gerente está relacionado a um departamento, e o departamento só pode ter um gerente.

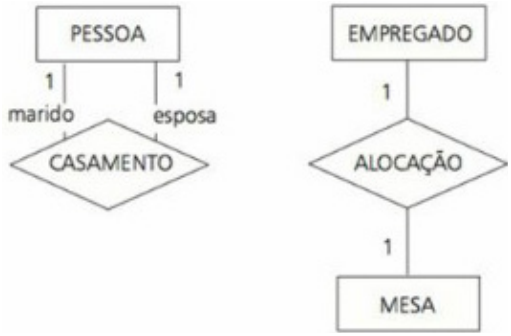
FIGURA 23 – RELACIONAMENTO 1..1



FONTE: O autor

Vamos imaginar um cenário onde há um cadastro currículo ou um diploma, ou um livro, sabe-se que cada pessoa pode ter apenas armazenado apenas um desses itens, e cada um desses itens só pertence a um único usuário cadastrado. Vamos ilustrar mais um pouco, imagine que o conjunto A da figura anterior seja sua família e que o conjunto B seja um cofre, cada pessoa só pode ter um único objeto e um objeto só pode pertencer a uma única pessoa.

FIGURA 24 – 1..1

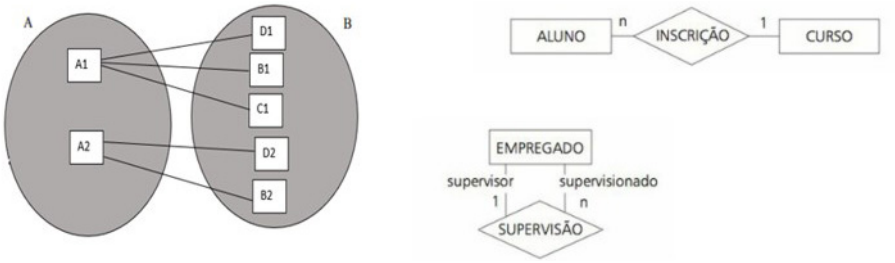


FONTE: Heuser (2009, p. 41)

Relacionamento 1..n ou 1..* (um para muitos):

Diferente do relacionamento anterior, onde há uma certa exclusividade de ambos os lados, nesse modelo, a exclusividade acontece somente em um lado. O relacionamento 1:n (um para muitos) ocorre com muitas vezes, por exemplos em situações de negócio. Imagine o cenário de uma universidade, onde uma disciplina possui vários alunos matriculados nela. A representação ficaria assim:

FIGURA 25 – 1..N

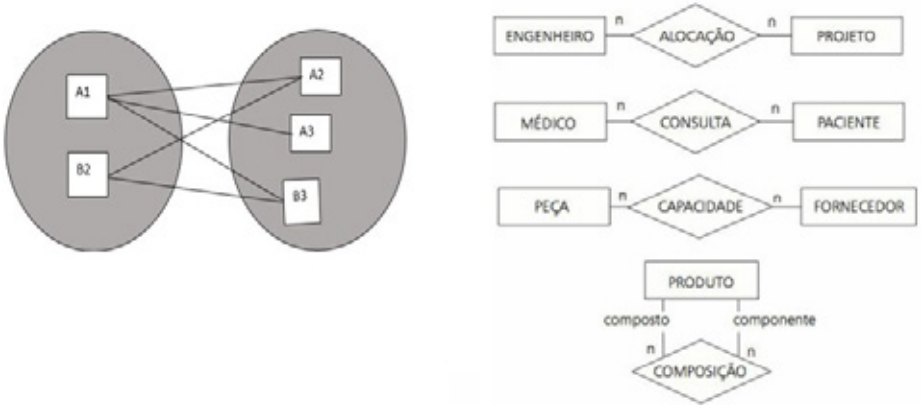


FONTE: O autor

Vejamos a figura a seguir onde uma das entidades envolvidas do conjunto A, por exemplo, pode referenciar várias unidades do conjunto B, porém, do outro lado cada uma das várias unidades referenciadas do conjunto B só pode estar ligada uma unidade do conjunto A. Um exemplo clássico é o do plano de saúde, onde o titular do plano de saúde, pode ter vários dependentes, mas cada dependente só pode estar ligado a um titular.

Relacionamento n..n ou *..* (muitos para muitos): neste tipo de relacionamento cada entidade, de ambos os lados, podeindicar várias unidades da outra. Vamos imaginar uma biblioteca por exemplo, um livro pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários livros. Dessa maneira, um objeto do tipo autor pode referenciar múltiplos objetos do tipo título, e vice-versa.

FIGURA 26 – N..N



FONTE: O autor

Esse tipo de relacionamento é bastante comum, veja alguns exemplos como os apresentados na figura a seguir. Onde um engenheiro pode fazer vários projetos, e um projeto pode ser desenvolvido por vários engenheiros. Como no caso do médico e paciente, onde um paciente pode consultar com vários médicos e um médico pode atender vários pacientes. Agora que já identificamos as entidades e a cardinalidade, vamos identificar os atributos.

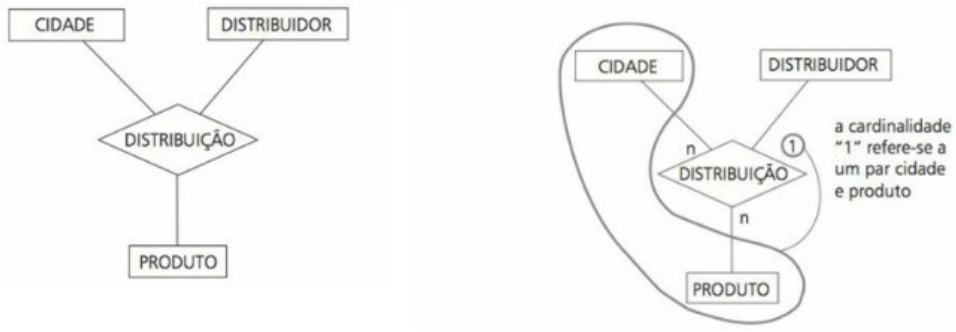
Relacionamento ternário: conforme as imagens apresentadas anteriormente são exemplos de relacionamentos binários. Mas a abordagem ER permite que sejam definidos relacionamentos de grau maior do que dois, como relacionamentos ternários, quaternários, entre outros. O DER da figura a seguir mostra um exemplo de um relacionamento ternário, segundo Heuser (2009, p. 30),

"Onde cada ocorrência do relacionamento DISTRIBUIÇÃO associa três ocorrências de entidade: um produto a ser distribuído, uma cidade na qual é feita a distribuição e um distribuidor".

No caso de relacionamentos de grau maior que dois, o conceito de cardinalidade de relacionamento é uma extensão não trivial do conceito de cardinalidade em relacionamentos binários. Lembre-se de que, em um relacionamento binário R entre duas entidades A e B, a cardinalidade máxima de A em R indica quantas ocorrências de B podem estar associadas a cada ocorrência de A. No caso de um relacionamento ternário, a cardinalidade refere-se a pares de entidades. Em um relacionamento R entre três entidades A, B e C, a cardinalidade máxima de A e B dentro de R indica quantas ocorrências de C podem estar associadas a um par de ocorrências de A e B.

Exemplificando, na figura, na linha que liga o retângulo representativo da entidade DISTRIBUIDOR ao losango representativo do relacionamento expressa que cada par de ocorrências (cidade, produto) está associado a no máximo um distribuidor. Significa que é concedida exclusividade de distribuição de um produto para um distribuidor em uma cidade.

FIGURA 27 – RELACIONAMENTO TERNÁRIO



FONTE: Adaptado de Heuser (2009, p. 31)

Atributos: para Heuser (2009) o conceito de atributo serve para associar informações a ocorrências de entidades ou de relacionamentos.

Até o momento falamos das entidades e do relacionamento entre elas, agora vamos apresentar o que compõe essas entidades, os atributos. Elas são as características que descrevem cada entidade dentro de determinado domínio ou minimundo. Nos exemplos das entidades, usamos banco, pessoas e produtos. Mas, todas elas têm caraterísticas que a identificam. Por exemplo, um cliente de um banco, possui nome, endereço, telefone e número da conta corrente. Já um cliente de uma biblioteca, terá atributos diferentes, pois estará em outro domínio.

E durante a identificação do minimundo que acontece na fase de análise de requisitos que começam a ser identificados os atributos relevantes de cada entidade naquele contextoespecífico, objetivando manter o modelo o mais simples possível e consequentemente armazenar apenas as informações que serão úteis futuramente. Vale ressaltar que é de extrema importância guardar apenas informações relevantes, por exemplo, faz sentido eu saber qual o nome dos padrinhos ou o número de sapato de um cliente de banco, e se o modelo for para uma igreja, é importante eu saber quem são os padrinhos? E para uma loja de sapato, é importante eu ter essa informação sobre meu cliente?

Segundo Devmedia (2014), os atributos podem ser classificados quanto a sua função da seguinte forma:

- Descritivos: representam característica intrínsecas de uma entidade, tais como nome ou cor.
- Nominativos: além de serem também descritivos, estes têm a função de definir e identificar um objeto. Nome, código, número são exemplos de atributos nominativos.
- Referenciais: representam a ligação de uma entidade com outra em um relacionamento. Por exemplo, uma venda possui o CPF do cliente, que a relaciona com a entidade cliente.

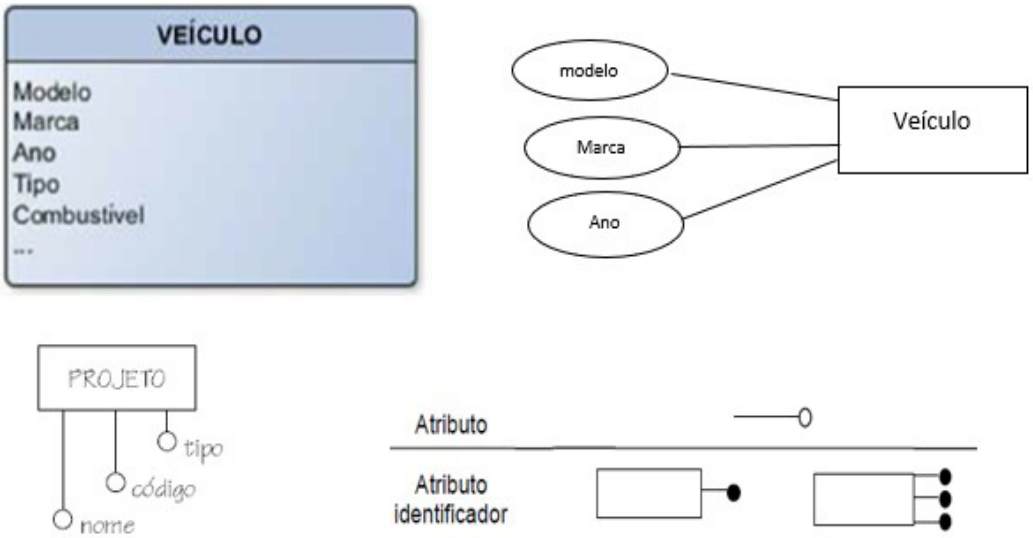
Quanto a sua estrutura, podemos ainda classificá-los como:

- Simples: um único atributo define uma característica da entidade. Exemplos: nome, peso.
- Compostos: para definir uma informação da entidade, são usados vários atributos. Por exemplo, o endereço pode ser composto por rua, número, bairro etc.

Além da classificação acima dos atributos, existem outras que logo iremos ver mais profundamente, que em alguns casos, alguns atributos também representam valores únicos que identificam a entidade dentro do domínio e não podem se repetir. Em um cadastro de clientes, por exemplo, esse atributo poderia ser o CPF. A estes chamamos de Chave Primária.

Podemos representar os atributos em um modelo de dados das seguintes formas:

FIGURA 28 – ATRIBUTOS



FONTE: O autor

LEITURA COMPLEMENTAR

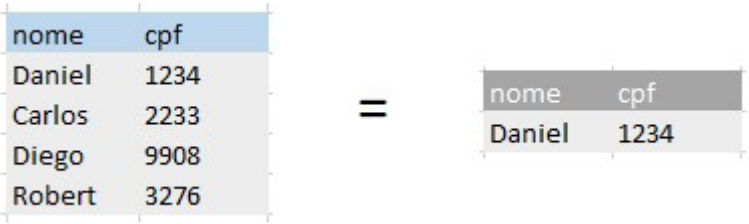
Neste artigo do site Space Programmer serão abordados os conceitos básicos de álgebra relacional. Esta que está por trás de consultas relacionadas a bancos de dados relacionais. A linguagem SQL foi criada a partir da álgebra relacional e por isso ter uma noção básica dos seus conceitos se torna muito importante.

Rocha, Anderson de Resende

Operações Básicas:

Seleção: σ : A seleção, como o próprio nome já diz, seleciona linhas, tuplas por meio de uma determinada condição.

Ex.: σnome="Daniel"(cliente)



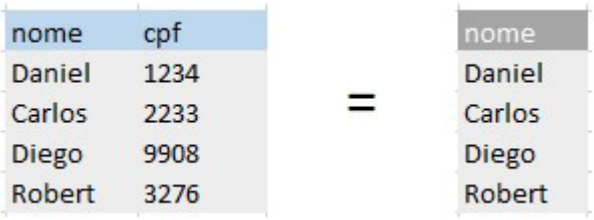
nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

=

nome	cpf
Daniel	1234

Projeção: π : A projeção é utilizada quando existe a necessidade de pegar somente colunas de interesse em uma relação, e não trabalhar com todas as colunas dessa relação.

Ex: πnome(cliente)



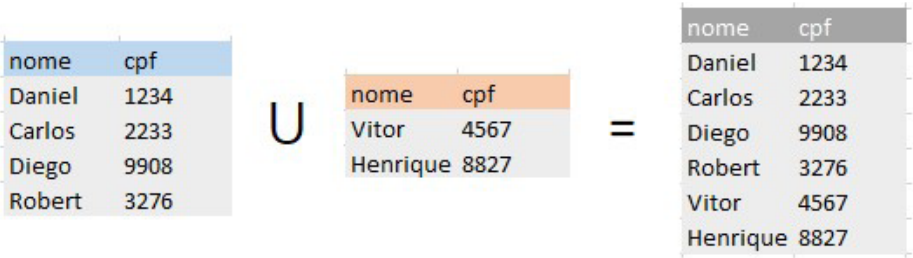
nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

=

nome
Daniel
Carlos
Diego
Robert

União: U: A união entre duas relações AUB, traz em uma nova relação C com todas as tuplas existentes em A e B, sem repetição de tupla.

Ex: (clientesA) U (clientesB)



nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

U

nome	cpf
Vitor	4567
Henrique	8827

=

nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276
Vitor	4567
Henrique	8827

Interseção: ∩ : A interseção de duas relações A∩B traz uma nova relação C contendo as tuplas, linhas em comum, ou seja, que existam nas duas relações.

Ex: (clientesA) ∩ (clientesC)

nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

∩

nome	cpf
Vitor	4567
Carlos	2233
Diego	9908

=

nome	cpf
Carlos	2233
Diego	9908

Diferença de conjuntos: − : A diferença entre duas relações A – B, traz uma nova relação C com tuplas que existem em A mas não existem em B, ou seja, que só existem em A. Do mesmo, equivale para B – A, traz uma nova relação com as tuplas que só existem em B, portanto, a ordem é importante.

Ex: (clientesA) − (clientesC)

nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

−

nome	cpf
Vitor	4567
Carlos	2233
Diego	9908

=

nome	cpf
Daniel	1234
Robert	3276

Produto cartesiano: × : O produto cartesiano entre duas relações A X B traz uma nova relação C que contém todos os campos, colunas que A e B contém, e a combinação de cada tupla de A com cada tupla em B.

Ex: (cliente) x (veiculo)

nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

X

carro	cpf
Bmw	1234
Ferrari	9908

=

nome	cpf	carro	cpf
Daniel	1234	Bmw	1234
Daniel	1234	Ferrari	9908
Carlos	2233	Bmw	1234
Carlos	2233	Ferrari	9908
Diego	9908	Bmw	1234
Diego	9908	Ferrari	9908
Robert	3276	Bmw	1234
Robert	3276	Ferrari	9908

Outras operações

Junção: ⋈ : Logo após a realização de um produto cartesiano em uma relação, geralmente existe a necessidade de se fazer uma seleção na qual os campos que estão em A e também estão em B sejam iguais, porque o produto cartesiano traz tuplas que não são de interesse. A junção é a operação que simplifica tudo isso. Ela realiza um produto cartesiano, depois uma seleção das tuplas de interesse e por fim uma projeção, para remoção de colunas duplicadas.

Ex: (cliente)⋈(veiculo)

nome	cpf
Daniel	1234
Carlos	2233
Diego	9908
Robert	3276

⋈

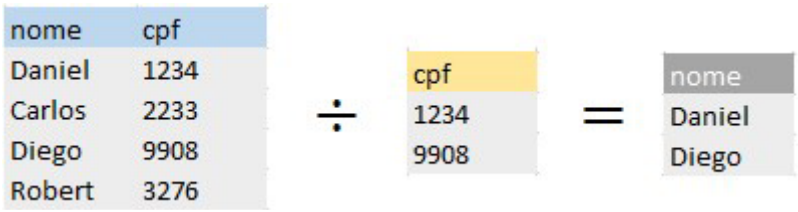
carro	cpf
Bmw	1234
Ferrari	9908

=

nome	cpf	carro
Daniel	1234	Bmw
Diego	9908	Ferrari

Divisão: ÷ : A operação de divisão entre duas relações A ÷ B traz uma nova relação C com todas as tuplas que possuem campos em comum nas duas relações.

Ex: (cliente) ÷ (cpfProcurado)



Renomeação: ϱ : A renomeação, de maneira intuitiva, permite renomear uma relação, chamar por mais de um nome, e também nomear uma operação em cima de uma relação, por exemplo, nomear o resultado da projeção em uma relação A.

Atribuição: ← EX: X ← (A × B)

A atribuição permite expressar consultas de uma maneira simples, de forma a ser possível trabalhar com estas.

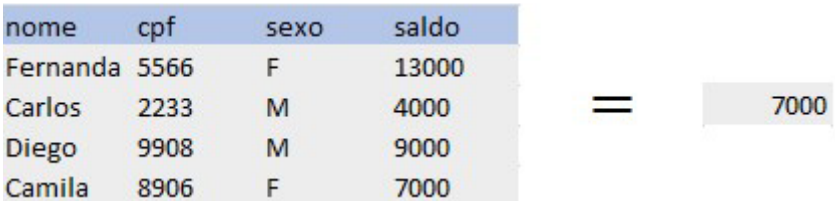
Funções e Operações agregadas

As operações agregadas foram criadas para simplificar as consultas às relações, de forma que algumas funções que são comumente utilizadas sejam facilmente acessadas.

avg: valor médio, min: valor mínimo , max: valor máximo , sum: soma dos valores , count: quantidade de um determinado valor. As operações agregadas são representadas por um *Γ* da seguinte maneira:

*Γ*operação_de_agregação (nome_da_coluna)(nome_da_relação)

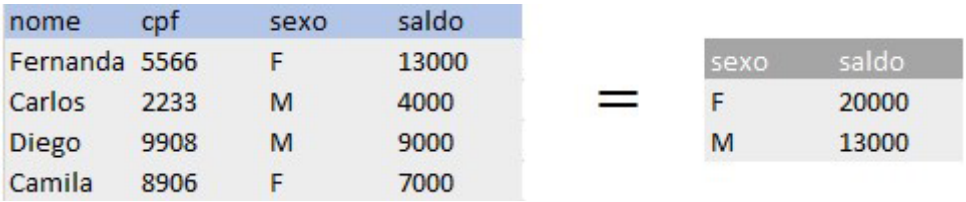
EX: *Γ*min (saldo)(Conta)



Você também pode aplicar essas operações de forma agrupada em relações:

(nome_do_campo_agrupado)*Γ*operação_de_agregação (nome_da_coluna)(nome_da_relação)

EX: (sexo)*Γ*sum (saldo)(Conta)



FONTE: <https://bit.ly/3avbxAw>. Acesso em: 10 ago. 2019.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- Podemos classificar as entidades em fortes, fracas e associativas, dependendo da sua utilização.
- Os relacionamentos entre as entidades, possuem uma cardinalidade máxima e mínima que se dividem em: Relacionamento 1..1 (um para um); Relacionamento 1..n ou 1..* (um para muitos), Relacionamento n..n ou *..* (muitos para muitos).
- Além dos relacionamentos entre duas tabelas, também temos o relacionamento entre três, chamado de ternário.
- Como são definidos os atributos.

Ficou alguma dúvida? Construímos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

AUTOATIVIDADE

1 O Modelo Entidade Relacionamento (também chamado Modelo ER, ou simplesmente MER), como o nome sugere, é um modelo conceitual utilizado na Engenharia de Software para descrever os objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como elas se relacionam entre si (relacionamentos). Considere as afirmações a seguir sobre a modelagem Entidade-Relacionamento.

I- Uma entidade pode não ter um valor aplicável para um atributo. Por exemplo, o atributo número do apartamento de um endereço só se aplica a endereços que estão em prédios de apartamento, e não a outros tipos de residências, como casas. Para tais situações, o valor especial NULL pode ser utilizado.
II- Para uma entidade de pessoa, o valor da idade pode ser determinado pela data atual e pelo valor da data de nascimento dessa pessoa. O atributo idade é chamado de atributo derivado e considerado derivável do atributo data de nascimento, que é chamado, por sua vez, de atributo armazenado.
III- Um atributo chamado de valor único ou monovalorado é aquele que pode ter um conjunto de valores para a mesma entidade. Por exemplo, para o atributo formação acadêmica, uma pessoa pode não ter formação acadêmica, outra pode ter e uma terceira pode ter duas ou mais formações.

FONTE: <https://bit.ly/2UNN9U8>. Acesso em: 10 ago. 2019.

Quais estão corretas?
a) ( ) Apenas I.
b) ( ) Apenas II.
c) ( ) Apenas III.
d) ( ) I e II.
e) ( ) I, II e III.

2 No mapeamento de um modelo entidade-relacionamento para um modelo relacional de banco de dados, o tipo de relacionamento que implica a criação de uma terceira tabela para onde serão transpostos as chaves primárias e os eventuais atributos das duas tabelas originais é denominado:

FONTE: <https://bit.ly/2vYvwZe>. Acesso em: 10 ago. 2019.

a) ( ) Relacionamento N:N.
b) ( ) Relacionamento 1:1.
c) ( ) Relacionamento 1:N.
d) ( ) Autorrelacionamento 1:N.
e) ( ) Relacionamento ternário.

3 Considere o modelo de dados a seguir, de uma clínica médica em que trabalham diversos médicos de diversas especialidades que prescrevem medicamentos e atendem a pacientes que podem estar acometidos com uma ou mais doenças.

Sobre este modelo, é correto afirmar que:

FONTE: <https://bit.ly/2V0mKmf>. Acesso em: 10 ago. 2019.

a) ( ) As entidades MÉDICO e PACIENTE estabelecem uma relação com cardinalidade 1:n.
b) ( ) Uma entidade MEDICAMENTO deverá ser adicionada ao modelo, relacionando-se com cardinalidade n:n diretamente com a entidade MÉDICO.
c) ( ) Falta a entidade MEDICAMENTO, que deverá estabelecer uma relação direta n:n com a entidade PACIENTE.
d) ( ) Se for adicionada a entidade MEDICAMENTO ao modelo ela deverá se relacionar com CONSULTA, que passa a ser uma entidade associativa.
e) ( ) As entidades MÉDICO e PACIENTE estabelecem uma relação com cardinalidade 1:1.

UNIDADE 2

ESTRUTURA DE UMA BASE DE DADOS

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você deverá ser capaz de:

- conhecer a origem do banco de dados relacional e sua fundação na álgebra relacional;
- identificar os tipos de dados e a sua capacidade de armazenamento;
- diferenciar as chaves primárias PK, chaves estrangeiras e chaves únicas;
- identificar os tipos de comandos: Data Manipulation Language (DML); Data Definition Language (DDL); Data Control Language (DCL); Transactional Control Language (TCL); Data Query Language (DQL).

PLANO DE ESTUDOS

Esta unidade está dividida em três tópicos. No decorrer da unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – ÁLGEBRA RELACIONAL E OPERAÇÕES RELACIONAIS

TÓPICO 2 – ESTRUTURA DE UMA BASE DE DADOS

TÓPICO 3 – COMANDOS DML

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

TÓPICO 1

ÁLGEBRA RELACIONAL E OPERAÇÕES RELACIONAIS

1 INTRODUÇÃO

Antes de iniciarmos nossos estudos, vamos conhecer e padronizar alguns termos que são utilizados, segundo a terminologia formal de modelo relacional. Uma linha também pode ser chamada de tupla. Já o cabeçalho da coluna é chamado de atributo, (conforme aprendemos na unidade anterior) e a tabela é chamada de relação.

Agora que já temos os conceitos definidos, vamos começar nossa unidade, conhecendo os princípios de uma consulta de um banco de dados relacional, que é da matemática. Conheceremos a álgebra relacional e as operações relacionais. Isso se justifica, pois, a maioria dos produtos desenvolvidos utilizam um banco de dados relacional, em que o usuário dá as instruções ao sistema para que o ele realize uma sequência de operações na base de dados para calcular e obter o resultado desejado. Em resumo, a álgebra relacional é uma forma de cálculo sobre conjuntos ou relações.

Vamos conhecer um pouco mais!

2 CONHECENDO ÁLGEBRA RELACIONAL E OPERAÇÕES RELACIONAIS

Vamos começar com o conceito de a Álgebra Relacional, segundo Devemedia (2008), que diz que ela é uma linguagem de consulta procedural, ou seja, são definidos os passos que serão executados. Ela é formal a qual a técnica utilizada é fundamental para a extração de dados de um banco de dados, além de ser um conjunto de operações, que utiliza como recurso de entrada uma ou mais relações, produzindo então, uma nova relação. Reforçando essa definição Ramarkrishnan (2011, p. 85).

> Cada consulta relacional descreve um procedimento passo a passo para computar a resposta desejada, baseado na ordem em que os operadores são aplicados na consulta. A natureza procedural da álgebra nos permite considerar uma expressão algébrica como uma receita, ou um plano, para avaliar uma consulta, e os sistemas relacionais realmente usam as expressões algébricas para representar os planos de avaliação das consultas.

Podemos resumir dizendo que a álgebra relacional é uma forma de cálculo sobre relações ou conjuntos. Neste tópico, iremos nos ater às operações fundamentais que são: projeção, seleção, produto cartesiano, união, diferença, subtração.

3 OPERADORES DE ÁLGEBRA RELACIONAL

Segundo Júnior (2016), podemos classificar em nove as operações ou operadores com a álgebra relacional. Eles podem ser agrupados em relação a sua origem como: fundamentais, derivados e especiais. Os fundamentais, caracterizam-se por que através dela qualquer expressão de consulta de dados é permitida, como; projeção, seleção, produto cartesiano, união, diferença e subtração. Já os derivados, derivam dos operadores fundamentais. Eles são definidos para facilitar a especificação de certos procedimentos: intersecção, junção (normal e natural), divisão. Já os especiais são os operadores que não se enquadram nos itens anteriores: Renomeação e alteração.

Júnior (2016) apresenta também a classificação quanto ao número de relações (tabelas), podendo ser classificadas como unárias, que são aquelas que operam em uma única tabela, que são: seleção, projeção, renomeação e alteração. E os que operam em duas tabelas, que são: união, intersecção, diferença, produto cartesiano, junção e divisão. Veremos mais detalhadamente nos exemplos apresentados.

Cada um dos noves operadores apresenta um símbolo que identifica sua função e operação dentro da expressão relacional, ou melhor dizendo dentro da fórmula que identifica e apresenta a operação relacional que está sendo realizada. O Quadro 1, apresentado a seguir, ilustra os símbolos representativos de cada operador.

QUADRO 1 – RESUMO DE OPERAÇÕES/OPERADORES EM ÁLGEBRA RELACIONAL

OPERAÇÃO	SÍMBOLO	SINTAXE
Projeção	π ("pi")	π <lista de campos> (Tabela)
Seleção/ Restrição	σ ("sigma")	σ <condição de seleção> (Tabela)
União	∪	(Tabela 1) ∪ (Tabela 2)
Interseção	∩	(Tabela 1) ∩ (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

FONTE: Júnior (2016, s.p.)

Para auxiliar nosso entendimento sobre o assunto e trabalho com os operadores, utilizaremos o modelo de dados (schema) proposto por Bezerra (2012) e apresentado no quadro a seguir:

QUADRO 2 – SCHEMA BASE

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

FONTE: Bezerra (2012, s.p.)

Após analisarmos o modelo proposto, começaremos definindo a projeção.

3.1 PROJEÇÃO (Π)

Para Bezerra (2012) a projeção pode ser entendida como uma operação que filtra as colunas de uma tabela de nosso banco de dados ou uma tabela resultante de outra operação relacional executada. Esse exemplo é uma operação unária, pois está sendo realizada em uma única tabela.

Sintaxe: π coluna1, coluna2,..., colunaN (Tabela).
Consulta: Qual o nome e data de admissão dos funcionários?
Comando: π Nomefunc, DtAdmissao (Funcionario).

Tomando como base o schema proposto por Bezerra (2012), vejamos o resultado da projeção, respondendo à consulta.

QUADRO 3 – RESULTADO PROJEÇÃO

NomeFunc	**DtAdmissão**
Luis Sampaio	10/08/2003
Carlos Pereira	02/03/2004
Jose Alves	25/05/2002
Luis Paulo Souza	10/12/2001
Marta Silveira	05/01/2002
Ana Luiza	12/01/2003
Pedro Sergio	29/06/2003
Larissa Silva	01/06/2002
Roberto Fernandes	15/10/2003
Sergio Nogueira	10/02/2000

FONTE: Adaptado de Bezerra (2012)

Seguindo com o nosso modelo, vejamos a seleção/restrição, que será muito utilizada durante nossa disciplina.

3.2 SELEÇÃO/ RESTRIÇÃO (**Σ**)

Essa é uma operação muito utilizada em um banco de dados relacional, pois segundo Júnior (2016), ela pode ser entendida como a operação que filtra, seleciona as linhas de uma tabela, realizando também uma projeção, e opera em um conjunto de dados, sendo, portando, uma operação unária.

Sintaxe: σ <condição de seleção ou predicado> (Tabela).
Consulta: Quais os funcionários de sexo masculino?
Comando: σ Sexo='M' (Funcionario).

QUADRO 4 – RESULTADO SELEÇÃO/RESTRIÇÃO

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
123	Pedro Sergio	29/06/2003	M	C6	D3
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

FONTE: Adaptado de Bezerra (2012)

Seguindo ainda com Bezerra (2012), se quisermos saber ainda, neste exemplo, somente o nome e a data de admissão dos funcionários então começamos a utilizar operações combinadas, pois deveremos fazer uma projeção desse resultado. E montaremos a seguinte consulta.

Consulta: Quais os nomes e data de admissão dos funcionários de sexo masculino? Comando: π Nomefunc, DtAdmissão (σ Sexo='M' (Funcionário)).

Para auxiliar o entendimento, podemos montar graficamente e teríamos a árvore de expressão e o resultado da consulta.

FIGURA 1 - ÁRVORE E EXPRESSÃO

FONTE: Adaptado de Bezerra (2012)

O resultado dessa seleção é apresentado a seguir, onde responderemos à seguinte pergunta: Quais os nomes e data de admissão dos funcionários de sexo masculino?

QUADRO 5 - RESULTADO SELEÇÃO/RESTRIÇÃO

NomeFunc	DtAdmissão
Luis Sampaio	10/08/2003
Carlos Pereira	02/03/2004
Jose Alves	23/05/2002
Luis Paulo Souza	10/12/2001
Pedro Sergio	29/06/2003
Roberto Fernandes	15/10/2003
Sergio Nogueira	10/02/2000

FONTE: Adaptado de Bezerra (2012)

Uma dica, é para que em consultas complexas, para facilitar o entendimento, construa a consulta graficamente antes de elaborar o comando em álgebra relacional.

3.2.1 Produto Cartesiano (X)

No produto cartesiano, Júnior (2016) utiliza a mesma notação de operação matemática de dois conjuntos, e obtém como resultado do produto cartesiano de duas tabelas uma terceira tabela, que contém as cominações possíveis entre os elementos das tabelas originais. O autor afirma que essa tabela resultante possui um número de colunas que é igual à soma do número de colunas das tabelas iniciais e um número de linhas igual ao produto do número de linhas das duas tabelas. Vejamos como ela funciona:

Sintaxe: (Tabela 1) X (Tabela 2).

Consulta: Trazer as informações dos funcionários e de seus cargos. Comando: Funcionarios X Cargos.

QUADRO 6 – RESULTADO PRODUTO CARTESIANO

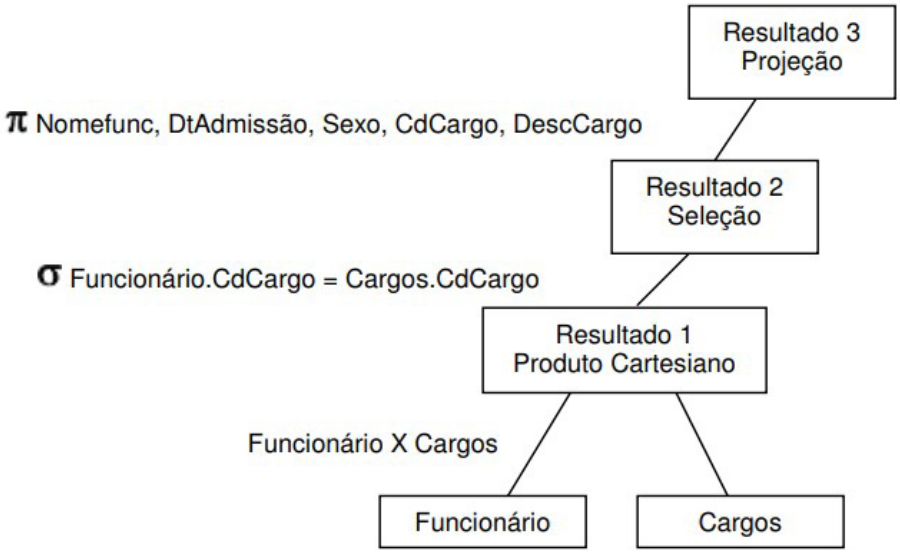
NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto	CdCargo	DescCargo	VlrSalario
101	Luis	10/08/2003	M	C3	D5	C1	Aux.Vendas	350,00
101	Luis	10/08/2003	M	C3	D5	C3	Vendedor	800,00
101	Luis	10/08/2003	M	C3	D5	C6	Diretor	2500,00
101	Luis	10/08/2003	M	C3	D5	C2	Vigia	400,00
101	Luis	10/08/2003	M	C3	D5	C5	Gerente	1000,00
101	Luis	10/08/2003	M	C3	D5	C4	Aux.Cobrança	250,00
104	Carlos	02/03/2004	M	C4	D6	C1	Aux.Vendas	350,00
104	Carlos	02/03/2004	M	C4	D6	C3	Vendedor	800,00
104	Carlos	02/03/2004	M	C4	D6	C6	Diretor	2500,00
104	Carlos	02/03/2004	M	C4	D6	C2	Vigia	400,00
104	Carlos	02/03/2004	M	C4	D6	C5	Gerente	1000,00
104	Carlos	02/03/2004	M	C4	D6	C4	Aux.Cobrança	250,00
.....	.....	.....	....	.....	.....	.....	.....	.....
.....	.....	.....	....	.....	.....	.....	.....	.....
22	Sergio	10/02/2000	M	C2	D4	C1	Aux.Cobrança	350,00
22	Sergio	10/02/2000	M	C2	D4	C3	Vendedor	800,00
22	Sergio	10/02/2000	M	C2	D4	C6	Diretor	2500,00
22	Sergio	10/02/2000	M	C2	D4	C2	Vigia	400,00
22	Sergio	10/02/2000	M	C2	D4	C5	Gerente	1000,00
22	Sergio	10/02/2000	M	C2	D4	C4	Aux.Cobrança	250,00

FONTE: Adaptado de Bezerra (2012)

Júnior (2016) traz um ponto de grande importância em relação ao produto cartesiano, pois ao abservamos que nossa operação é um produto cartesiano, teremos como resultado o produto das linhas das duas tabelas. Vale reforçar que isso em um projeto lógico, não está correto. Como resultado lógico, para nossa consulta as linhas que realmente nos interessam seriam as em que os campos CdCargo sejam iguais das duas tabelas, ou seja, Funcionário.CdCargo = Cargos. CdCargo, aplicamos então uma seleção no resultado:

Consulta: trazer o nome, data de admissão código do cargo e descrição do cargo dos funcionários.

FIGURA 2 – ÁRVORE DE EXPRESSÃO PRODUTO CARTESIANO



FONTE: Adaptado de Bezerra (2012)

A seguir, vejamos os dados corretamente selecionado, retornando apenas o solicitado na consulta: trazer o nome, data admissão, cód. do cargo e descrição, dos funcionários do sexo masculino, onde os campos CdCargo sejam iguais das duas tabelas, ou seja, Funcionário.CdCargo = Cargos.CdCargo.

QUADRO 7 – RESULTADO PRODUTO CARTESIANO CORRETO

NomeFunc	DtAdmissão	Sexo	CdCargo	Desccargo
Luis Sampaio	10/08/2003	**M**	C3	Vendedor
Carlos Pereira	02/03/2004	**M**	C4	Aux.Cobrança
Jose Alves	23/05/2002	**M**	C5	Gerente
Luis Paulo Souza	10/12/2001	**M**	C3	Vendedor
Pedro Sergio	29/06/2003	**M**	C7	Diretor
Roberto Fernandes	15/10/2003	**M**	C3	Vendedor
Sergio Nogueira	10/02/2000	**M**	C2	Vigia

FONTE: Adaptado de Bezerra (2012)

Podemos perceber que o resultado é mais coerente e não apresenta linhas repetidas, conforme podemos observar no Quadro 6, onde para cada funcionário, ele trazia todos os cargos. Já no Quadro 7, o resultado é apenas o cargo do funcionário.

3.2.2 União (∪)

Por utilizar duas tabelas, a união é uma operação binária, e como resultado cria uma tabela a partir de duas outras tabelas compatíveis, levando as linhas comuns e não comuns a ambas. Júnior (2016) reforça que as informações duplicadas aparecerão somente uma vez no resultado. São consideradas tabelas União Compatíveis, aquelas tabelas cuja quantidade, disposição e domínio dos atributos/ campos sejam os mesmos.

Sintaxe: (Tabela 1) ∪ (Tabela 2)

Comando: Imaginemos um caso onde temos a tabela de Contas e a de Empréstimos e queremos saber quais clientes possuem Conta ou Empréstimo, para isso realizamos as projeções respectivas para que tenhamos somente o campo NomeCliente de cada tabela e realizamos a união desse resultado. Vejamos o exemplo proposto por Bezerra (2012).

QUADRO 8 – EXEMPLO UNIÃO

Projeção 1
NomeCliente
Luis Sampaio
Carlos Pereira
Jose Alves

∪

Projeção 2
NomeCliente
Luis Sampaio
Luis Sampaio

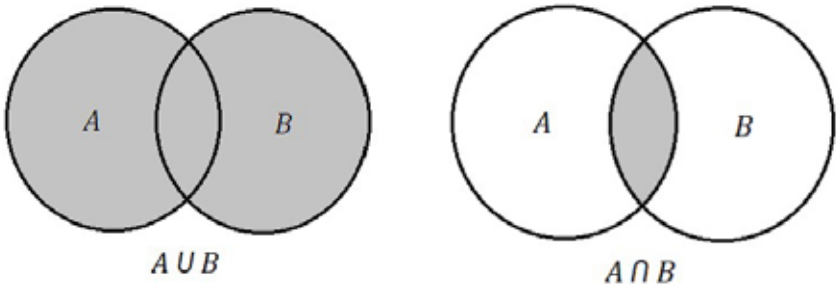
=

Resultado
NomeCliente
Luis Sampaio
Carlos Pereira
Jose Alves

FONTE: Adaptado de Bezerra (2012)

Como podemos observar no quadro, o resultado apresenta somente uma ocorrência, não duplicando os dados, diferente da intersecção, que veremos a seguir. Vejamos uma imagem para relembramos o conceito de união e intersecção, da teoria dos conjuntos.

FIGURA 3 – UNIÃO E INTERSECÇÃO



FONTE: Adaptado de Lessa (2018)

3.2.3 Intersecção (∩)

A intersecção também é uma operação binária, ou seja, gera uma tabela a partir de duas outras tabelas levando sem repetição as linhas, que pertençam a ambas as tabelas presentes na operação.

Sintaxe: (Tabela 1) ∩ (Tabela 2).

Comando: imaginemos um caso onde temos a tabela de Contas e a de Empréstimos e queremos saber quais clientes possuem Conta e Empréstimo, para isso realizamos as projeções respectivas para que tenhamos somente o campo NomeCliente de cada tabela e realizamos a intersecção desse resultado.

QUADRO 9 – INTERSECÇÃO

Projeção 1
NomeCliente
Luis Sampaio
Carlos Pereira
Jose Alves

∩

Projeção 2
NomeCliente
Luis Sampaio
Luis Sampaio

=

Resultado
NomeCliente
Luis Sampaio

FONTE: Adaptado de Bezerra (2012)

3.2.4 Diferença (-)

Para Júnior (20160 a diferença nos permite encontrar as linhas que estão em uma tabela, mas não estão em outra, a diferença entre elas. Como se fizéssemos a seguinte expressão: Tabela 1 – Tabela 2 = Tabela 3. Onde a Tabela 3 resulta em uma tabela que contém todas as linhas que estão na Tabela 1 e não estão na Tabela 2. Observamos que Tabela 1 – Tabela 2 (Exemplo1) é diferente de Tabela 2 – Tabela 1 (Exemplo2).

Sintaxe: (Tabela 1) – (Tabela 2)

Comando: imaginemos um caso onde temos a tabela de Contas e a de Empréstimos e queremos saber quais clientes possuem uma Conta, mas não tenham um Empréstimo, para isso realizamos as projeções respectivas para que tenhamos somente o campo NomeCliente de cada tabela e realizamos a diferença desse resultado. Vejamos os dois exemplos, propostos por Bezerra (2012).

QUADRO 10 – DIFERENÇA EXEMPLO 1

Projeção 1
NomeCliente
Luis Sampaio
Carlos Pereira
Jose Alves

\-

Projeção 2
NomeCliente
Luis Sampaio
Luis Sampaio

=

Resultado
NomeCliente
Carlos Pereira
Jose Alves

FONTE: Adaptado de Bezerra (2012)

Como podemos observar no exemplo 1 na tabela de resultado, temos as linhas que estão na projeção 1 e não estão na projeção 2. Vejamos outro exemplo, invertendo as posições das projeções.

QUADRO 11 – DIFERENÇAS EXEMPLO 2

Projeção 1
NomeCliente
Luis Sampaio
Carlos Pereira
Jose Alves

\-

Projeção 2
NomeCliente
Luis Sampaio
Luis Sampaio

=

Resultado (vazio)

FONTE: Adaptado de Bezerra (2012)

Neste novo cenário, não teremos nenhuma linha de retorno, pois todas as linhas da projeção 2, encontram-se na projeção 1.

3.2.5 Junção (⋈)

Na operação junção, ela interage com o modelo relacional, ou seja, trabalha com o modelo de relações entre tabelas realizando um produto cartesiano, combinando as linhas e somando as colunas de duas tabelas, só que partindo de campos comuns de ambas para realizar essa "seleção relacional", conforme Bezerra (2012). Vale reforçar que essa operação possui uma condição onde se colocam os campos das tabelas que estão sendo usados para se efetivar a junção. Chamamos essa junção de junção com predicado, conforme apresentado na Sintaxe 1.

Sintaxe1 (Junção com Predicado): (Tabela 1) ⋈ <condição de junção> (Tabela 2).

Comando: π Nomefunc, DtAdmissão, NumCargo
σ (Sexo='M' (Funcionário ⋈ Funcionário.CdCargo = Cargo.CdCargo Cargos))

Outro tipo de junção é a junção natural, que vemos na sintaxe 2, nela não há especificação de condição sendo usado para isso todas as colunas comuns às duas tabelas. As colunas resultantes são a soma das colunas das duas tabelas sem a repetição das colunas idênticas (aparecerão uma vez somente). Não deve ser empregada quando se deseja associar duas tabelas apenas por um ou alguns dos seus atributos idênticos, caso isso seja feito os resultados são imprevisíveis, segundo Bezerra (2012).

Sintaxe2 (Junção Natural): (Tabela 1)⋈(Tabela 2).

Ex: (Tabela 1)⋈<condição de junção> (Tabela 2).

Consulta: trazer o nome, data de admissão e cargo dos funcionários do sexo masculino.

Comando: π Nomefunc, DtAdmissão, NumCargo (σ Sexo='M' (Funcionário ⋈ Cargos))

QUADRO 12 – JUNÇÃO

NomeFunc	DtAdmissão	DescCargo
Luis Sampaio	10/08/2003	Vendedor
Carlos Pereira	02/03/2004	Aux.Cobrança
Jose Alves	23/05/2002	Gerente
Luis Paulo Souza	10/12/2001	Vendedor
Pedro Sergio	29/06/2003	Diretor
Roberto Fernandes	15/10/2003	Vendedor
Sergio Nogueira	10/02/2000	Vigia

FONTE: Adaptado de Bezerra (2012)

Independendente do tipo de junção realizada, o resultado será o mesmo.

3.2.6 Divisão (÷)

Essa operação, segundo Bezerra (2012), tem como resultado a projeção de todos os elementos da primeira tabela que se relacionam com todos os elementos da segunda tabela. Essa operação também pode ser obtida através de outras operações de álgebra relacional. Sintaxe: (Tabela 1) ÷ (Tabela 2). Vamos analisar a imagem a seguir:

FIGURA 4 – DIVISÃO

Tabela Equipe

NumReg	Cd Projeto
101	Projeto 1
104	Projeto 1
134	Projeto 1
101	Projeto 2
104	Projeto 2
101	Projeto 3

Tabela Projeto

CdProjeto	Descrição
Projeto 1	Suporte
Projeto 2	Manutenção
Projeto 3	Desenvolvimento

Consulta: Quais os funcionários que trabalham em todos os projetos?
Comando: (π NumReg, CdProjeto (EQUIPE)) + (π CdProjeto (PROJETOS))

Projeção 1

NumReg	Cd Projeto
101	**Projeto 1**
104	Projeto 1
134	Projeto 1
101	**Projeto 2**
104	Projeto 2
101	**Projeto 3**

+

Projeção 2

CdProjeto
Projeto 1
Projeto 2
Projeto 3

Resultado

NumReg
101

FONTE: Adaptado de Bezerra (2012)

Este cenário é diferente do que vimos até o momento, estamos utilizando para facilitar o entendimento. Neste caso, o operador dividendo é a tabela resultante da projeção 1, e o operando divisor é a tabela resultante da projeção 2, e a tabela de resultado é o quociente da operação.

3.2.7 Renomeação (ρ)

Como o nome sugere, esta é a operação que renomeia uma tabela. Segundo Macoratti (2013), ela também é uma operação unária primitiva, redefine o nome de uma tabela em um determinado contexto. Muito útil para autorrelacionamentos, quando precisamos fazer a junção de uma tabela com ela mesma, e, nesse caso, cada versão da tabela precisa receber um nome diferente da outra.

Sintaxe: ρ Nome(Tabela)
Exemplo: Funcionairo2(Funcionario)
Estou

3.2.8 Atribuição (←)

É utilizada para simplificar comandos muito extensos definindo então passos de comando. Macoratti (2013) afirma que essa função permite que o conteúdo de uma relação seja atribuído (colocado) em uma variável especial, oferecendo a possibilidade de um tratamento até certo ponto algorítmico para algumas sequências de operações. Atribui-se a relação resultante de uma operação à direita de, a uma variável temporária, à esquerda, que poderá ser utilizada em relações subsequentes.

Sintaxe: Variável ← Tabela Exemplo: Nomes_Func ← π Nomefunc(Funcionario) .

Finalizamos este tópico com as principais funções, mas não são as únicas.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- A projeção representada por π, e é entendida como uma operação que filtra as colunas de uma tabela.

- A seleção/restrição representada por σ, que é a operação que seleciona e filtra as linhas de uma tabela.

- O produto cartesiano representado por X, gera como resultado do produto cartesiano de duas tabelas uma terceira tabela, que contém as cominações possíveis entre os elementos das tabelas originais.

- A união cria uma terceira tabela a partir de duas outras tabelas compatíveis, levando as linhas comuns e não comuns a ambas. Seu símbolo é $\cup$.

- A intersecção só retornará às linhas que pertençam as duas tabelas. É representado por $\cap$.

- Diferença, nos permite encontrar as linhas que estão em uma tabela, mas não estão em outra, a diferença entre elas. Seu símbolo é -.

- Junção, trabalha com o modelo de relações entre tabelas realizando um produto cartesiano, combinando as linhas e somando as colunas de duas tabelas, só que partindo de campos comuns de ambas para realizar essa "seleção relacional", seu símbolo é $\bowtie$.

- A divisão é representada por $\div$, ela tem como resultado a projeção de todos os elementos da primeira tabela que se relacionam com todos os elementos da segunda tabela.

- A renomeação possui o seguinte símbolo ϱ, e ela renomeia uma tabela.

- A atribuição é a relação resultante de uma operação à direita de, a uma variável temporária, à esquerda, a qual poderá ser utilizada em relações subsequentes. Seu símbolo é $\leftarrow$.

AUTOATIVIDADE

1 Como vimos, a álgebra relacional possui um conjunto de operadores, cada um deles toma uma ou várias relações como entrada e produz uma nova relação como saída. Com base nisso, considere o seguinte comando escrito em álgebra relacional e selecione a opção correta que esse comando resulta. σproduto = caderno(Materiais)

a) ( )Apresenta apenas do atributo produto, da tabela Materiais, considerando os registros em que o valor do atributo produto seja igual a "caderno".
b) ( ) Seleciona todos atributos da tabela Materiais, para os registros em que o valor do atributo produto seja igual a "caderno".
c) ( ) Resulta na exclusão dos registros da tabela Materiais, apenas no caso de o valor do atributo produto ser igual a "caderno".
d) ( ) Irá inserir um registro na tabela Materiais, com o valor do atributo produto igual a "caderno".
e) ( ) Apresenta todos atributos da tabela Materiais, substituindo-se todos os valores do atributo produto pelo valor "caderno".

FONTE: Adaptado de <https://www.qconcursos.com/questoes-de-concursos/disciplinas/tecnologia-da-informacao-banco-de-dados/algebra-relacional/questoes>. Acesso em: 15 fev. 2020.

2 A Álgebra Relacional é uma linguagem de consulta formal, porém procedimental, ou seja, o usuário dá as instruções ao sistema para que o mesmo realize uma sequência de operações na base de dados para calcular o resultado desejado. Ela serviu de base para o desenvolvimento da linguagem SQL. Com base nas operações propostas pela Álgebra Relacional, selecione a opção correta:

a) ( ) Seleção, Restrição e Remodelação.
b) ( ) Variável, Seleção e Produto Cartesiano.
c) ( ) União, Renomeação e Junção.
d) ( ) Intervenção, Junção e Adição.

3 Com base no seguinte cenário de uma empresa, composta pela tabela Funcionário, responda, qual a opção correta que apresenta a seleção do funcionário com o nome João.

Funcionário

NOME	CPF	IDADE	Sexo
Antônio	999888777-88	25	M
João	888999777-55	43	M
Carla	777999666-77	33	F

a) ( ) σ **nome = 'João' (Funcionário).**
b) ( ) π (Nome) (Pessoa).
c) ( ) $\bowtie$ **nome = 'João' (Funcionário).**
d) ( ) **nome** $\cup$ **'João' (Funcionário).**

4 Continuando com o modelo apresentado na questão 3, responda qual o comando para apresentar somente o nome e a idade dos funcionários que tenham menos de 30 anos.

a) ( ) π nome, sexo, idade (π idade < 30) (Funcionário).
b) ( ) σ nome, idade (σ idade < 30).
c) ( ) π nome, idade (σ idade < 30) (João).
d) ( ) π nome, idade (σ idade < 30) (Funcionário).

5 Tomando como base a seguinte tabela, onde CPF é uma chave primária. Selecione a opção que apresente qual é a expressão, em álgebra relacional, que retorna os Nomes dos empregados com salário inferior a 3200 e que trabalhem no departamento de Compras?

CPF	NOME	Sobrenome	departamento	Salário
999888777-88	Antônio	Liberato	Vendas	3500
888999777-55	João	Caetano	Compras	2800
777999666-77	Carla	Amarante	Compras	3100
888999444-55	Sara	Jane	Compras	3300

EMPREGADO (CPF, Nome, sobrenome, Departamento, Salario)

a) ( ) σNome (σDepartamento = Compras; Valor < 3300).
b) ( ) πNome (σDepartamento = Compras AND Salario < 3200 (EMPREGADO)).
c) ( ) σNome (πSalario < 3200 AND Departamento = Compras).
d) ( ) σNome (πSalario < 3200 AND Departamento = Compras (EMPREGADO)).
e) ( ) πNome (πDepartamento = Compras; πSalario < 4000 (EMPREGADO))

TÓPICO 2

ESTRUTURA DE UMA BASE DE DADOS

1 INTRODUÇÃO

Agora já conhecemos a álgebra relacional e como podemos obter o resultado através da junção de tabelas. Vamos começar a conhecer como construir uma base de dados relacional. Começaremos conhecendo as chaves e restrições, pois ela garante a integridade da base de dados.

2 CHAVES E RESTRIÇÕES

Na unidade anterior, conhecemos o modelo lógico, e as chaves correspondem aos atributos identificadores, e dão uma identificação a cada ocorrência de instância em uma tabela, e garantem que elas não se repitam, tornando-as únicas. Mas, essa não é sua única função, elas ajudam a estabelecer o relacionamento entre as tabelas de um banco de dados relacional. Existem três tipos de chaves em um banco de dados relacional: chave primária, chave alternativa e a chave estrangeira, segundo Lehmkuhl (2013). Em seguida veremos mais detalhes referentes aos tipos de chaves e suas respectivas características.

Chave primária: é a chave que vai garantir a integridade dos dados, evitando assim a duplicidade de dados na coluna. Uma chave primária é uma coluna ou uma combinação de colunas cujos valores distinguem uma linha das demais dentro de uma tabela. Caso eu esteja armazenando os dados de uma pessoa física ou jurídica, eu posso usar o campo CPF/CNPJ para como chave primária, ou posso criar um campo novo, para garantir essa integridade. Conforme Fanderuff (2003, p. 47),

> Chave primária (primary key – pk): é a principal chave de acesso a uma tabela. A criação dessa chave faz com que, automaticamente, a tabela seja ordenada por essa chave e que não seja permitida a duplicidade em seu valor. É um conjunto de um ou mais campos que, tomados coletivamente, permite-nos identificar unicamente uma linha em uma tabela (integridade da entidade). Ou seja, o valor da PK de uma ocorrência nunca se repetirá nas demais ocorrências da mesma tabela. Uma PK é escolhida dentre várias chaves candidatas, que podem estar presentas na relação. As chaves candidatas não selecionadas são ditas chaves alterativas. A cada chave alternativa de ser associada uma restrição de integridade que garanta que seus valores sejam únicos (unique).

Seguindo o modelo apresentado no tópico anterior, vejamos uma chave primária, na figura a seguir. Perceba que os valores da coluna CdCargo não se repetem.

QUADRO 13 – CHAVE PRIMÁRIA

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux.Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

FONTE: Adaptado de Bezerra (2012)

Chave estrangeira: podemos dizer que é a chave estrangeira ou *foreign key* (FK), o mecanismo que permite a implementação de relacionamentos em um banco de dados relacional.

Segundo Lehmkuhl (2013), ela pode ser uma coluna, ou pode ser composta por um conjunto de colunas que se referem necessariamente a uma chave primária de outra tabela. Em alguns casos, ela pode referenciar a própria tabela, no caso de recursividade. Este relacionamento garante a integridade dos dados relacionados, pois apenas serão permitidos valores que atendam ao relacionamento.

QUADRO 14 – CHAVE ESTRANGEIRA

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

FONTE: Adaptado de Bezerra (2012)

Segundo Heuser (2001), para a criação de uma chave estrangeria, algumas restrições que ser garantidas ao executar operações de alterações do DB.

- Na inclusão: quando ocorrer a inclusão de uma linha na tabela que contém a chave estrangeira, deve ser garantido que o valor da chave estrangeira apareça na coluna da chave primária referenciada.
- Na alteração: quando ocorrer uma alteração de valor da chave estrangeira deve ser garantido que o novo valor de uma chave estrangeira apareça na coluna da chave primária referenciada.
- Na exclusão: caso haja a exclusão de uma linha da tabela que contém a chave primária referenciada pela chave estrangeira. Ou seja, não pode excluir uma linha da tabela Cargo, caso tenha algum funcionário neste departamento.

Chave única: em algumas literaturas, como a de Heuser (2001), a chave única (Unique key- UK) pode ser chamada de chave alternativa. Muitas vezes, o valor de uma coluna não pode ser repetido, mas também não é uma coluna que possa ser utilizada como chave primária, pois seu valor em alguns casos pode estar nulo. Por isso, além da chave primária, que evita a duplicidade, a chave única também evita dados duplicados, mas permite que eles estejam nulos. Um exemplo de chave única, é o e-mail.

2.1 SQL (*STRUCTURED QUERY LANGUAGE*)

Vamos conhecer as categorias de comando de um banco de dados. A SQL (Structured Query Language) é uma linguagem padrão utilizada para a comunicação com um banco de dados relacional. Ela foi inicialmente desenvolvida pela IBM nos anos 70, como uma interface para um sistema de banco de dados relacional denominado Sistema R. atualmente.

Segundo Cabeça (2009), ela é a linguagem mais usada pelos Sistemas Gerenciador de Banco de Dados Relacionais (SGBDRs) comerciais e tem sido bastante modificada pelos diversos fornecedores, o que resultou em diferentes tipos de comandos, muitas vezes incompatíveis entre os fornecedores. O que levou a necessidade de uma padronização, que hoje é a ANSI SQL-3. Ela é descrita em cinco documentos inter-relacionados visando à cobertura de todos os aspectos da linguagem.

Este padrão contempla, segundo Cabeça (2009, p. 11):

> (i) linguagem para a definição de dados (DDL – Data Definition Language), para a descrição das relações do banco de dados;
> (ii) linguagem de manipulação de banco de dados (DML – Data Manipulation Language), para especificar a manipulação do estado da base de dados;
> (iii) linguagem de consulta de dados (DQL – Data Query Language), para especificar a consulta ao banco de dados;
> (iv) linguagem de controle dos dados (DCL – Data Control Language), para especificar o controle de acesso aos dados contidos no banco de dados;
> (v) comandos de administração de dados; e controle de transações [ANSI – SQL3].

Basicamente, os comandos se dividem em quatro grandes grupos, que veremos no decorrer do tópico. São eles:

- Data Manipulation Language (DML).
- Data Definition Language (DDL).
- Data Control Language (DCL).
- Transactional Control Language (TCL).
- Data Query Language (DQL).

O grupo DQL, é composto pelo comando SELECT, que como veremos mais à frente, recupera linhas do banco de dados e permite a seleção de uma ou várias linhas ou colunas de uma ou várias tabelas.

- *Data Manipulation language* (DML)

Os comandos do grupo DML (*data Manipulation language*) ou Linguagem de Manipulação de Dados, são o grupo de instruções usada nas consultas e modificações dos dados armazenados nas tabelas do banco de dados. Ou seja, eles mantêm, alteram e excluem os dados, ou seja, os conteúdos das tabelas. São eles:

- INSERT -> Instrução utilizada para inserir dados a uma ou mais tabelas no banco de dados.
- UPDATE -> Instrução utilizada para atualizar dados de uma ou mais tabelas no banco de dados.
- DELETE -> Instrução utilizada para excluir dados de uma ou mais tabelas no banco de dados.
- MERGE -> Realiza operações de inserção, atualização ou exclusão em uma tabela de destino com base nos resultados da junção com a tabela de origem.

- *Data definition language* (DDL)

O outro grande grupo, também de grande importância, é o DDL. Os comandos DML só podem ser executados, graças às estruturas criadas no DDL (*Data definition language*) ou linguagem de Definição de Dados). É um conjunto de instruções usado para criar e modificar as estruturas dos objetos armazenados no banco de dados. Pense nele como as paredes de uma casa. Alguns Exemplos:

- ALTER -> Instrução utilizada para modificar a definição de entidades existentes. Use ALTER TABLE para adicionar uma nova coluna a uma tabela ou use ALTER ATABASE para definir opções do banco de dados.
- CREATE -> Instrução utilizada para definir novas entidades. Use CREATE TABLE para adicionar uma nova tabela em um banco de dados.
- DROP -> Instrução utilizada para remover entidades existentes. Use DROP TABLE para remover uma tabela de um banco de dados.
- TRUNCATE TABLE -> Remove todas as linhas de uma tabela sem registrar as exclusões de linhas individuais.

Semelhante aos comandos DDL, que são normalmente responsabilidade do administrador do banco de dados, os comandos do tipo DCL (*Data Control Language*) ou Linguagem de Controle de Dados – são usados para controle de acesso e gerenciamento de permissões para usuários em no banco de dados. Com eles, pode facilmente permitir ou negar algumas ações para usuários nas tabelas ou registros (segurança de nível de linha). Aqui estamos colocando novamente apenas alguns para ilustrar, pois eles são utilizados com banco multiusuários, quando há necessidade de dar ou limitar acesso dos usuários.

Alguns exemplos como GRANT, que atribui privilégios de acesso do usuário a objetos do banco de dados. O REVOKE remove os privilégios de acesso aos objetos obtidos com o comando GRANT.

Existe uma lista imensa de comandos DDL e DCL, normalmente o DBA é o responsável por utilizar. No nosso livro didático, focaremos mais nos comandos básicos DDL e nos comandos DML.

O grupo de TCL (*Transaction control language*) ou Linguagem de Controle de Transações são usados para gerenciar as mudanças feitas por instruções DML. Ele permite que as declarações a serem agrupadas em transações lógicas. Ou seja, depois que eu fizer alguma alteração nos meus dados, eu preciso confirmar ou desistir. São eles o COMMIT, que é usado para salvar permanentemente qualquer transação no banco de dados. E o ROLLBACK, que desfaz a ação e restaura o banco de dados para o último estado commited.

Antes de começarmos a efetivamente a criação do nosso banco de dados, precisamos conhecer os tipos de dados. Já começamos a ver quando fizemos o modelo lógico. Os modelos apresentados são os utilizados pelo banco de dados Oracle, mas a diferença entre os outros bancos é mínima.

2.2 TIPOS DE DADOS

Para que possamos criar a nossa base de dados, é necessário conhecer os tipos de dados disponíveis. No quadro a seguir, apresentamos um resumo, dos tipos mais utilizados.

Conheça todos s tipos de dados em: https://bit.ly/2x3vhge.

QUADRO 15 – RESUMO TIPO DE DADOS

Datatype	Descrição
CHAR	Um campo de caractere de comprimento fixo com até 2000 *bytes* de comprimento.
NCHAR	Um campo de comprimento fixo para conjuntos de caracteres *multibyte*. O tamanho máximo é de 2000 caracteres ou 2000 *bytes*, dependendo do conjunto de caracteres.
VARCHAR2	Um campo de caractere de tamanho variável, com até 4000 caracteres de comprimento.
NVARCHAR2	Um campo de comprimento variável para conjuntos de caracteres *multibytes*. O tamanho máximo é de 4000 caracteres ou 4000 *bytes*, dependendo do conjunto de caracteres.
DATE	Um campo de comprimento fixo de 7 *bytes* usados para armazenar todas as datas. O tempo é armazenado como parte da data. Quando consultada, a data estará no formato DD-Mon-YY, como em 15-OCT-99 para 15 de outubro de 1999, a menos que o parâmetro NLS_DATE_FORMAT seja definido no INIT. ORA.
NUMBER	Uma coluna de número de comprimento variável. Os valores permitidos são zero e números positivos e negativos. Os valores NUMBER geralmente são armazenados em 4 ou menos *bytes*.
LONG	Um campo de comprimento variável com até 2GB de comprimento.
RAW	Um campo de comprimento variável usado para dados binários de até 2000 *bytes* de comprimento.
LONG RAW	Um campo de comprimento variável usado para os dados binários com até 2GB de comprimento.
BLOB	*Binary large object*, com até 4 GB de comprimento.
CLOB	*Character large object*, com até 4 GB de comprimento.
NCLOB	Datatype CLOB para conjuntos de caracteres *multibyte* e até 4GB de comprimento.
BFILE	Arquivo externo binário, o tamanho é limitado pelo sistema operacional.
ROWID	Dados binários que representam o RowID. O valor será de 10 *bytes*.

FONTE: Alessio (2013, p. 70)

Vamos observar alguns detalhes, por exemplo, no que se refere à diferença entre os tipos VARCHAR2 e CHAR, que é a capacidade total de armazenamento. No varchar2 o tamanho de um campo varia de acordo com seu conteúdo, o valor declarado é apenas um limitador, já quando se usa o tipo CHAR o valor declarado é fixo independente do seu conteúdo.

Vamos imaginar que estamos criando uma coluna chamada NOME do tipo Varchar2(20), e nela vamos armazenar o conteúdo 'Ana', serão usadas apenas três posições. Já se tivéssemos definido como Char(20), todas as posições de memória seriam utilizadas. Por isso, quando temos valores variáveis, e não sabemos o tamanho total, é mais interessante utilizar o tipo VARCHAR2. Em casos onde eu sei o tamanho final, como por exemplo Unidade federativa que só ocupa dois caracteres, podemos usar CHAR (2).

Veremos utilização desses tipos, quando formos criar uma tabela ou uma procedure/função.

É interessante que você instale o Oracle e o SQL developer para praticar, é grátis. Acesse https://www.oracle.com/br/downloads/. Você deve fazer o cadastro no site da Oracle, para ter acesso grátis às ferramentas.

- COMANDO DDL

Conheceremos os comandos DDL (Data Definition Language) ou Linguagem de Definição de Dados, define a estrutura dos dados e tabelas. Os comandos DDL mais comuns são CREATE, ALTER, DROP, RENAME e TRUNCATE.

Esses comandos serão muito importantes, pois a partir deles iremos criar, manter e caso seja necessário apagar tabelas ou linhas em nosso banco. Vale ressaltar que esses comandos não precisam ser confirmados via *commit*, e não podem ser desfeitos via *rollback*, portanto muita atenção ao executar.

- CRIANDO TABELA

As instruções do tipo DDL, são um subconjunto de instruções SQL, que são utilizadas para criar, modificar ou remover estruturas de dados Oracle. Temos que ter cuidado, pois são ação e imediata no banco, sem a necessidade de confirmação, como ocorre nos comandos DML.

- CREATE – Comando utilizado para criar um BANCO (CREATE DB) e uma TABELA (CREATE TABLE).

- CREATE DB – Cria um banco de dados. Veja um exemplo de utilização:

- CREATE DATABASE database_name que são: Create, alter e drop.

Semelhante as definições de variáveis, temos algumas regras para a nomeação de colunas e tabelas:

- Devem sempre começar com uma letra;
- Devem ter de 1 a 30 caracteres. (Sempre usar nomes significativos);
- Devem conter apenas caracteres de A .. Z, a ..z, 0..9, _ (sublinhado), $ e # (caracteres legais, mas não é aconselhado o uso);
- Não deve duplicar o nome de outro objeto do mesmo usuário;
- Não deve ser nenhuma palavra reservada;
- Não é *case sensitive*, ou seja, não faz diferença entre maiúscula e minúscula. Como, por exemplo, Empregado ou emprEGAdo.

Vamos começar os comandos DDL com a criação de tabelas, devemos estar atentos que antes de fazer os comandos no banco de dados Oracle, é necessário que tenhamos os privilégios para realizar.

- CREATE TABLE [SCHEMA.] table (column datatype [default exp] [, …]);

Essa sintaxe apresentada é o modelo que você encontrará nas documentações, mas não se preocupe, iremos detalhar cada um dos itens.

- **Schema** – é o mesmo nome do proprietário (usuário que logou no sistema).
- **Table** – é o nome da tabela que está sendo criada no banco de dados.
- **DEFAULT** exp – é o valor padrão, caso não seja informado outro no insert, por exemplo letra 'S' na coluna de aluno ativo Sim ou não.
- **Column** – é o nome da coluna.
- **Datatype** – é o tipo de dados e o comprimento da coluna, conforme declarado no tópico anterior, Quadro 15.

Na figura a seguir, já estamos utilizando a ferramenta Oracle SQL Developer, onde digitamos o comando DDL para a criação da tabela. Observe que criamos um padrão de nomenclatura, onde os campos numéricos começam com o prefixo **NRO** e os campos textos com os prefixos (**NOM e DSC**), na coluna do tipo date também. Após a digitação do código, executamos o comando, clicando na seta em verde, conforme sinalizado na Figura A. Na coluna de data da criação, deixamos o valor de *sysdate* (data do dia na máquina) como valor default, caso não seja informado na hora do *insert*.

Feito isso, e caso não haja erros, uma mensagem é apresentada conforme a Figura B. E para verificarmos na base, a nossa tabela, podemos observar a Figura C. Outra maneira de verificar a estrutura da tabela no banco é através do comando DESC ou DESCIBRE.

desc departamento;
describe departamento;

FIGURA 5 – CRIANDO TABELA

FONTE: O autor

Em alguns casos queremos criar uma tabela com as mesmas colunas de uma já existente, ou igual ao resultado de uma coluna, para isso, podemos criar uma tabela utilizando uma subconsulta: Ex.: Create table NOME [(coluna, coluna...)] AS subquery; É muito importante que você pratique, você pode usar outras ferramentas on-line.

- Chave primária

Mas como vimos anteriormente, as tabelas têm algumas restrições que devem ser deifinidas durante sua criação, para garantia da sua integridade e unicidade. Vejamos as sintaxes para a definições de restrições:

```
SINTAXE : CREATE TABLE [schema].table
(column datatype [DEFAULT expr]
[column_constraint],
…
[table_constraint] [,…]);
SINTAXE da restrição no nível coluna :
    Column [constraint constraint_name] constraint_type,

SINTAXE da restrição no nível tabela :
    Column [constraint constraint_name] constraint_type
    (column, …),
```

Complementando a definição já realizada, agora temos os campos column_constraint e table_constraint, onde o column_constraint, que se referem à restrição de integridade como parte da definição da coluna, e a table_constraint, é uma restrição de integridade como parte da definição da tabela. Vejamos na prática:

```
Exemplo da restrição no nível coluna :
CREATE TABLE empregado (
cod_empregado number(6) CONSTRAINT emp_cod_emp_pk PRIMARY
KEY,
nom_empregado varchar2(50),
vlr_salario number(8,2));
Exemplo da restrição no nível tabela :
CREATE TABLE empregado (
cod_empregado number(6),
nom_empregado varchar2(50),
vlr_salario number(8,2),
 CONSTRAINT emp_cod_emp_pk PRIMARY KEY (cod_empregado));
```

Podemos criar a restrição de primary key, no momento em que estamos criando a coluna, onde não há necessidade de criamos um nome para a chave, ou podemos criar em nível de tabela, onde temos que indicar qual o nome da PK e a qual coluna se refere.

O banco de dados ORACLE trabalha com METADADOS, ou seja, as estruturas das tabelas, colunas e chaves são armazenas em tabelas. Esse controle sobre essas estruturas, é atribuição do DBA (Administrador de banco de dados).

2.3 RESTRIÇÃO NOT NULL

Além da restrição de chave primária, as vezes precisamos que mesmo não sendo a chave primária, que seu preenchimento seja obrigatório, como o nome ou a data de nascimento de uma pessoa, em um cadastro de uma empresa. Para resolver essa necessidade, temos a restrição NOT NULL.

Essa restrição, garante que a coluna não tenha algum valor nulo. Vale ressaltar a diferença entre valor nulo e zero. Um valor nulo é quando não foi informado nada, agora se o usário digitar espaços ou informar algum outro caracter, a coluna não será nula. Infelizmente não temos como garantir a qualidade dos valores, apenas que eles serão ou não populados. As colunas que são definidas como chave primária já são not null.

```
CREATE TABLE empregado (
cod_empregado number(6),
nom_empregado varchar2(50) not null,
vlr_salario number(8,2) not null,
 CONSTRAINT emp_cod_emp_pk PRIMARY KEY (cod_empregado));
```

2.4 RESTRIÇÃO CHAVE ÚNICA

Diferente da chave primária em que a coluna é obrigatória, e diferente da coluna not null onde podemos ter os valores repetidos, a chave única (unique key), não é obrigatória (exceto se você definir a coluna como not null), mas quando for informado um valor, ele deve ser diferente dos demais.

```
Sintaxe:
CREATE TABLE empregado (
cod_empregado number(6),
nom_empregado varchar2(50) not null,
vlr_salario number(8,2) not null,
dsc_email varchar2(50),
 CONSTRAINT emp_cod_emp_pk PRIMARY KEY (cod_empregado),
CONSTRAINT dsc_email_uk UNIQUE (dsc_email));
```

```
Ou
CREATE TABLE empregado (
cod_empregado number(6) CONSTRAINT emp_cod_emp_pk PRIMARY KEY,
nom_empregado varchar2(50),
vlr_salario number(8,2),
dsc_email varchar2(50) CONSTRAINT dsc_email_uk UNIQUE );
```

2.5 RESTRIÇÃO CHAVE ESTRANGEIRA

A restrição de chave estrageira (*FOREING KEY*), ou integridade referencial, designa uma ou mais colunas e estabelece o relacionamente com a chave primária em uma tabela diferente, ou na mesma tabela. O valor da chave estrangeira, sempre deve ser igual ao valor da tabela pai, ou nulo. Onde a tabela pai, neste exemplo, é a REGIAO, que se relaciona com a tabela PAIS. A criação das chaves pode ser definida via coluna, ou via tabela, semelhante ao que vimos até o momento. Observe que na criação da chave estrangeira (FK), referenciamos (REFERENCE) a tabela com quem ela se relaciona e o nome da coluna.

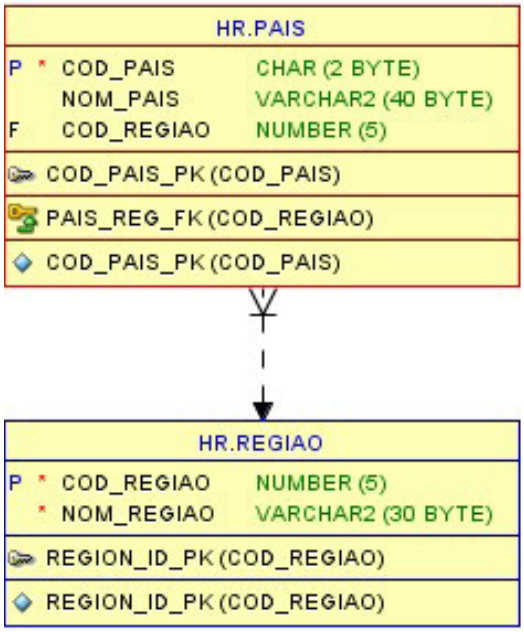
Criação da tabela pai:

```
CREATE TABLE REGIAO
   (cod_regiao NUMBER(5) CONSTRAINT REGION_ID_PK PRIMARY
KEY,
   nom_regiao VARCHAR2(30)not null);
```

Criação da tabela filho, onde referencia a tabela pai.

```
CREATE TABLE pais
  (cod_pais  CHAR(2) CONSTRAINT cod_pais_pk primary key,
        nom_pais  VARCHAR2(40),
        cod_regiao      NUMBER(5)   CONSTRAINT   PAIS_REG_FK
REFERENCES REGIAO(COD_REGIAO) );
```

FIGURA 6 – CHAVE ESTRANGEIRA



FONTE: O autor

2.6 RESTRIÇÃO CHECK

Vimos até o momento várias restrições, para garantir a integridade dos dados. A restrição CHECK é uma garantia de que somente os valores permitidos serão inseridos. Por exemplo, evitar que um salário seja informado com zero.

Sintaxe:

```
CREATE TABLE empregado (
cod_empregado number(6),
nom_empregado varchar2(50),
vlr_salario number(8,2) CONSTRAINT emp_min_sal CHECK(vlr_salario > 0) ,
 CONSTRAINT emp_cod_emp_pk PRIMARY KEY (cod_empregado) );
```

Ou

```
alter table nome_tabela add constraint nome_check check(coluna
condição);
```

exemplo: alter table TABELA_TEMPO add constraint nome_ck check(sal_anual > 0);

2.7 ALTERANDO E APAGANDO

Em alguns casos, precisamos modificar as estruturas criadas, ou até mesmo apagá-las. Esses comandos servem para alterar tanto as estruturas das tabelas, quanto as restrições. Começaremos com a ALTER TABLE, esse comando usado para adicionar, excluir ou modificar as colunas de uma tabela existente.

Sintaxe para adiciona coluna:

ALTER TABLE table_name ADD column_name datatype;

Exemplo: **ALTER TABLE empregado ADD dsc_email varchar2(50) CONSTRAINT dsc_email_uk UNIQUE** ;

Podemos também utilizar o ALTER TABLe, para incluir as chaves primárias e estrangeiras. Neste exemplo estamos utilizando uma chave primária composta.

Sintaxe:
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n);

Exemplo: **ALTER TABLE empregado**
ADD CONSTRAINT empregado_pk PRIMARY KEY (cod_empregado, nom_empregado);

Semelhante a chave primária, podemos também alterar e adicionar a chave estrangeira.

Sintaxe:
ALTER TABLE table_name
ADD FOREIGN KEY (column_name) REFERENCES table_name (column_name);

Exemplo: **ALTER TABLE empregado**
ADD FOREIGN KEY (cod_departamento) REFERENCES Departamento (cod_departamento);

Como já mencionamos, existe casos em que precisamos mudar uma tabela, excluindo uma coluna.

Sintaxe para para excluir uma coluna:

ALTER TABLE table_name DROP COLUMN column_name;

Exemplo: **alter table EMPREGADO drop column DSC_EMAIL cascade constraints;**

Ao excluirmos uma coluna, é importante também excluirmos as constraints. Caso haja a necessidade de exluir a tabela inteira, utilizamos o comando DROP.

Lembrando que os comandos DDL, não exigem confirmação, uma vez executados, não há como voltar atrás.

O DROP é o comando utilizado para exclusão da estrutura no banco.

Sintaxe: **DROP TABLE table_name;**

Exemplo: **DROP TABLE EMPREGADO;**

Em algumas situações, apenas renomear já resolve o problema, para isso existe o comando RENAME.

Sintaxe: **ALTER TABLE NOME_ANTIGO rename to nome_novo;**

Exemplo: ALTER TABLE EMPREGADO rename to FUNCIONARIO;

Caso a alteração, não seja de estrutura, mas sim os conteúdos de uma tabela, pode-se limpar os dados e manter a estrutura, truncando a tabela. Desta maneira a tabela se mantem, mas os dados são apagados.

Sintaxe: **TRUNCATE TABLE nome_tabela;**
Exemplo: **Truncate table empregados;**

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- A chave primeira garante a integridade e unicidade dos dados.
- A chave estrangeira ou *Foreign Key* (FK) permite a implementação de relacionamentos em um banco de dados relacional.
- A chave estrangeira é também conhecida como chave alternativa.
- Os comandos do grupo DML (*data Manipulation Language*), ou Linguagem de Manipulação de Dados, são as instruções usadas nas consultas e modificações dos dados armazenados nas tabelas do banco de dados.
- Os comandos do grupo DDL (*Data Definition Language*), ou linguagem de Definição de Dados), é um conjunto de instruções usado para criar e modificar as estruturas dos objetos armazenados no banco de dados.

AUTOATIVIDADE

Usando o modelo de dados do Tópico 2 desta unidade, vamos criar a nossa base de dados.

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

FONTE: Adaptado de Bezerra (2012)

1 Com base na imagem, crie as tres tabelas com as respectivas chaves primárias.

2 Com base nas chaves pimárias, faça a criação das chaves estrangeiras.

3 Crie uma restrição, para que a coluna sexo, receba somente os valores definidos.

TÓPICO 3

COMANDOS DML

1 INTRODUÇÃO

Agora que já aprendemos como criar os comandos DDL e já temos uma pequena base de dados, vamos conhecer os comandos DML (Data Manipulation Language). Através desses comandos, vamos inserir dados na nossa base, manipular e excluir dados. Diferente das instruções DDL que são executadas automaticamente no banco, sem a necessidade de confirmação, nos comandos DML há a necessidade de confirmação (Commit) ou de desistência (Roolback).

Uma transação, é um conjunto de instruções DML. Imagine que sua base de dados seja sua instituição bancária, onde você faz várias operações. Para garantir que o saldo sempre esteja correto, caso haja algum erro, o banco volta para a última transação commitada, e as demais transações serão desfeitas. Vamos começar!

2 INSERINDO

As inserções devem ser realizadas, uma linha por vez. Vamos utilizar a tabela criada anteriormente, e a sintaxe para inserção é:

Sintaxe: INSERT INTO table [(column[,colum..])] VALUES (value[, value]);

Existem duas maneiras de realizar a inserção, a primeira é quando definimos as colunas para a inserção. Neste caso, se não quisermos inserir os valores não obrigatórios, basta não indicar as colunas. Já a segunda é quando seguimos a sequência definida na criação. Vejamos o primeiro exemplo, tomando como base a seguinte tabela.

```
CREATE TABLE EMPREGADO
  (COD_EMPREGADO NUMBER(6),
          NOM_EMPREGADO VARCHAR2(50),
          VLR_SALARIO NUMBER(8,2) not null,
          DSC_EMAIL VARCHAR2(50),
        CONSTRAINT EMP_MIN_SAL CHECK (vlr_salario > 0),
        CONSTRAINT EMP_COD_EMP_PK PRIMARY KEY (COD_EM
PREGADO),
        CONSTRAINT DSC_EMAIL_UK UNIQUE (DSC_EMAIL));
```

Exemplo de Insert com definição de colunas:

INSERT INTO empregado(COD_EMPREGADO,VLR_SALARIO) VALUES(2, 1589);

Nesta primeira forma de insert, nós definimos quais são as colunas que queremos inserir, e quais valores queremos informar. Nós decidimos somente inserir a coluna que é a chave primária e a coluna que é not null.

A outra forma de inserção, nós não informamos quais as colunas vamos inserir, e devemos seguir a estrutura definida na criação. Novamente, nós não precisamos informar os valores que não são obrigatórios e podemos atribuir nulo. Essa prática não é recomenda, pois, a base não estará coerente, imagine um cadastro de funcionário, sem o nome dele.

Exemplo de insert sem definição de colunas e usando null:

INSERT INTO empregado VALUES (2,null, 1589, null);

Observe que no insert a seguir, as colunas do tipo texto devem estar entre aspas simples. E a coluna salário, como foi definida como um NUMER(8,2), tem a capacidade de armazenamento de 8 dígitos no total, sendo 6 inteiros e 2 decimais (999999.99). Outro ponto que deve ser observado é a pontuação, neste caso está sendo usado o padrão americano, em que as casas decimais são separadas por ponto.

Exemplo de insert:

INSERT INTO empregado VALUES(2,' Antonio Chaves', 1589.85, 'antonnio.chaves@email.com.br');

Caso eu tente inserir outra linha, apenas alterando o valor do salário para zero, com base no que definimos como restrições, a ferramenta irá apresentar os seguintes erros:

INSERT INTO empregado VALUES(2,' Antonio Chaves', 0, 'antonnio.chaves@email.com.br');

Relatório de erros -

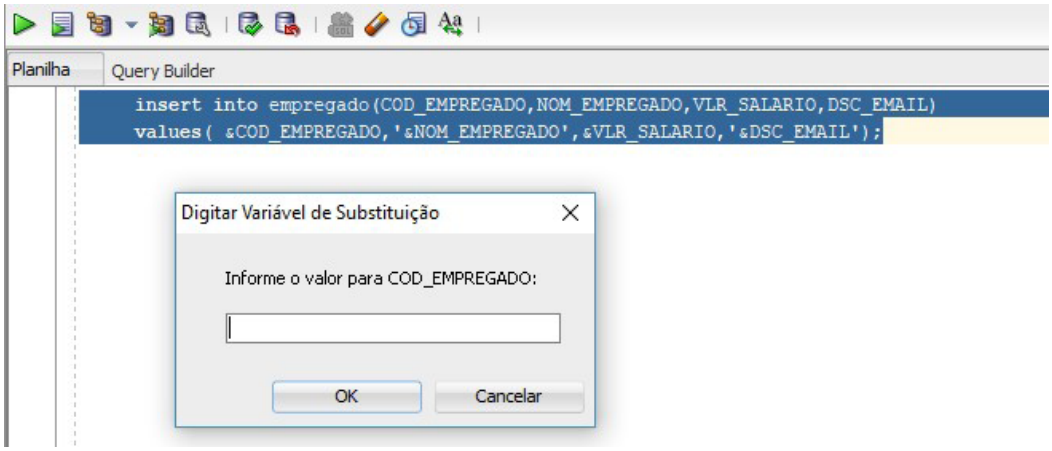
ORA-02290: restrição de verificação (HR.EMP_MIN_SAL) violada (indicando que a restrição de salário foi violada).

ORA-00001: restrição exclusiva (HR.EMP_COD_EMP_PK) violada (indicando que já existe um empregado com a chave primária informada)

ORA-00001: restrição exclusiva (HR.DSC_EMAIL_UK) violada (indica que o email informado já existe).

Uma maneira de não ter que cada vez editar o código utilizando uma variável precedida pelo símbolo &. Lembrando que campos do tipo data e alfanuméricos, devem estar entre aspas simples, veja o modelo. Para cada variável que foi precedida pelo &, uma janela será aberta para a digitação, conforme figura a seguir.

FIGURA 7 – INSERINDO COM VARIÁVEIS



FONTE: O autor

Após fazer a inserção no banco, podemos alterar seu conteúdo através do comando update.

a. Alterando

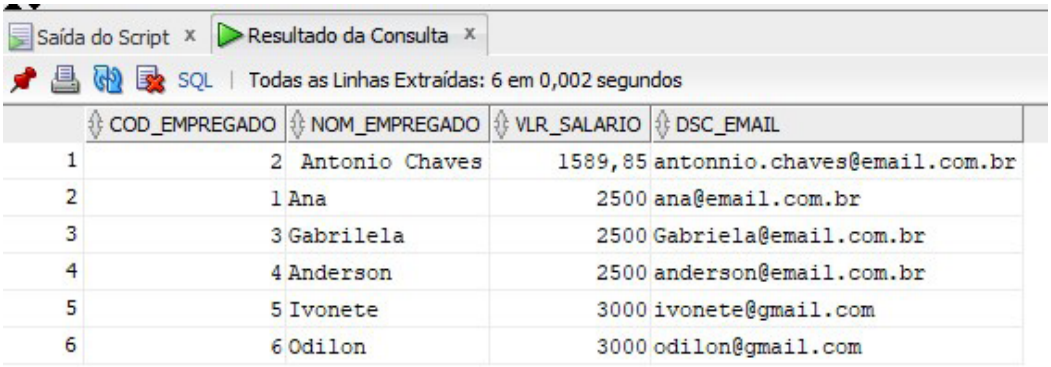
Fazer uma alteração nos valores/conteúdo de uma ou mais colunas em um banco de dados, pode ser perigoso se não for feita corretamente. Por isso, ter conhecimento de como funciona a instrução, com certeza, minimiza em muito os problemas.

Sintaxe: **UPDATE table**
SET column = value [, column = value, …]
[where condition];

O comando WHERE permite limitar a nossa pesquisa, ou seja, informamos que condições queremos, ou seja, os registros a serem retornados podem ser restritos através da inclusão da cláusula WHERE.

Na instrução acima, a opção "condition" identifica as linhas a serem atualizadas e é composta pelo nome das colunas, expressões, subconsultas e operadores de comparação. Para evitar problemas, uma dica é usar a coluna que contém a chave primária na cláusula WHERE, assim saberemos exatamente qual é a linha que será atualizada. Imagine a seguinte base de dados, conforme a figura a seguir.

FIGURA 8 – BASE DADOS



	COD_EMPREGADO	NOM_EMPREGADO	VLR_SALARIO	DSC_EMAIL
1	2	Antonio Chaves	1589,85	antonnio.chaves@email.com.br
2	1	Ana	2500	ana@email.com.br
3	3	Gabrilela	2500	Gabriela@email.com.br
4	4	Anderson	2500	anderson@email.com.br
5	5	Ivonete	3000	ivonete@gmail.com
6	6	Odilon	3000	odilon@gmail.com

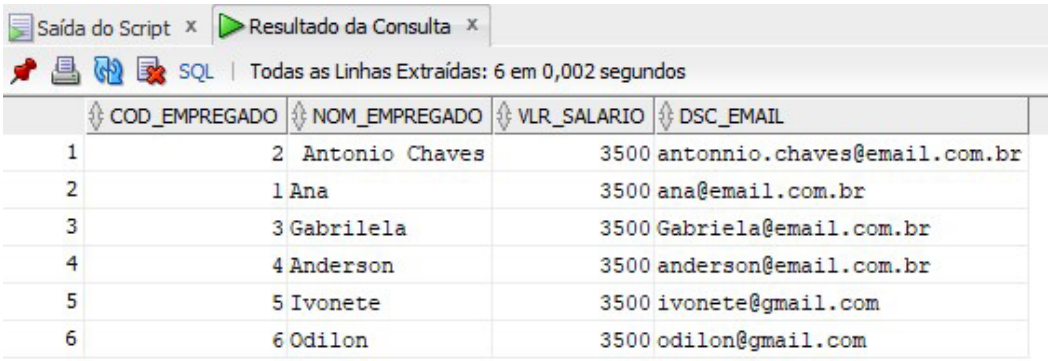
FONTE: O autor

Se ao analisar a base de dados, percebermos que alguma informação não está coerente, podemos alterá-la. Imagine que o salário do empregado código 2 Antonio, está incorreto e executasse a seguinte instrução:

Exemplo: UPDATE empregado
SET vlr_salario = 3500;

Veja como ficaria a base, onde podemos observar a nossa base, após a nossa alteração, como não definimos, ou seja, nós não restringimos qual linha deveria ser atualizada, todas a linhas tiveram seu valor alterado. Como o update é uma instrução DML, podemos voltar atrás e desfazer utilizando o comando rollback.

FIGURA 9 – SEM WHERE



	COD_EMPREGADO	NOM_EMPREGADO	VLR_SALARIO	DSC_EMAIL
1	2	Antonio Chaves	3500	antonnio.chaves@email.com.br
2	1	Ana	3500	ana@email.com.br
3	3	Gabrilela	3500	Gabriela@email.com.br
4	4	Anderson	3500	anderson@email.com.br
5	5	Ivonete	3500	ivonete@gmail.com
6	6	Odilon	3500	odilon@gmail.com

FONTE: O autor

Como é um comando DML, ele permite que seja desfeito, através do comando rollback. Agora faremos novamente a intrução, informando qual é a regra de atualização que deve ser colocada no comando WHERE.

```
update empregado
set vlr_salario = 3500
where cod_empregado = 2;
```

Ao identificar, na cláusula WHERE qual linha queremos alterar, o sistema só atualiza a linha indicada. O mesmo acontece no caso de excluir uma linha. Por isso é muito importante a sua utilização.

b. Apagando

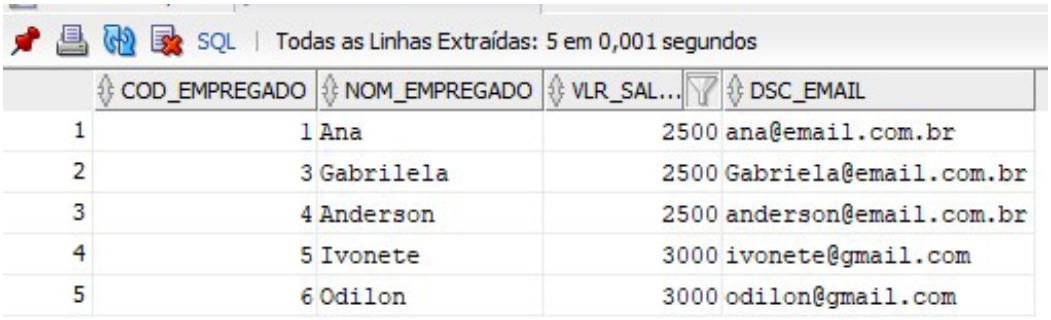
Se por algum motivo, nós resolvermos excluir uma linha na nossa base de dados, nós também devemos especificar qual é a linha, conforme sintaxe apresentada.

Sintaxe: **DELETE [FROM] table**
[WHERE condition]

Semelhante ao comando update, a cláusula WHERE é opcional, mas se ela não for informada, TODOS os registros da tabela serão apagados.

Exemplo: **delete from empregado**
where cod_empregado = 2;

FIGURA 10 – APAGANDO


SQL | Todas as Linhas Extraídas: 5 em 0,001 segundos

	COD_EMPREGADO	NOM_EMPREGADO	VLR_SAL...	DSC_EMAIL
1	1	Ana	2500	ana@email.com.br
2	3	Gabrilela	2500	Gabriela@email.com.br
3	4	Anderson	2500	anderson@email.com.br
4	5	Ivonete	3000	ivonete@gmail.com
5	6	Odilon	3000	odilon@gmail.com

FONTE: O autor

Caso a intenção seja mesmo remover todas as linhas, além do delete sem WHERE, pode-se usar o comando TRUNCATE. Ele remove TODAS as linhas, mas mantém a estrutura da tabela.

Sintaxe: **TRUNCATE TABLE table_name;**
Exemplo: **TRUNCATE TABLE empregado**.

c. MERGE

O comando Merge permite mesclar linha de uma tabela em outra. Este comando permite combinar várias operações para reduzir a complexidade de inserções e atualizações. Com esse comando é possível combinar as instruções de INSERT, UPDATE e DELETE, em uma única instrução, segundo Costa (2015).

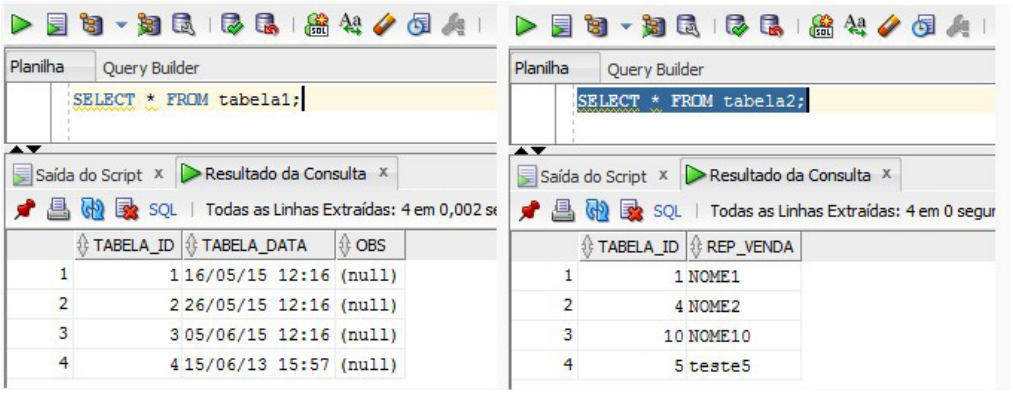
Vejamos a sintaxe:
MERGE INTO tabela_destino
USING tabela_origem | subquery
ON condicao
WHEN MATCHED THEN UPDATE SET coluna = expressao | DEFAULT
where_clause
DELETE where_clause
WHEN NOT MATCHED THEN INSERT (colunas)
VALUES(valores | DEFAULT)
where_clause
WHERE condicao;

Como apresentado por Costa (2015), inicialmente pode parecer bem confuso entender essa sintaxe. A **tabela destino** é a tabela que irá receber um INSERT ou um UPDATE, já a **tabela origem** é onde irá buscar os dados. **On condicao** é justamente o que irá ocorrer com cada linha, se será atualizada ou inserida de acordo se a condição foi satisfeita ou não. Opcionalmente ela poderá ser excluída também. Vamos ver um exemplo agora.

```
CREATE TABLE tabela1(
tabela_id NUMBER,
tabela_data DATE,
obs VARCHAR2(20));

CREATE TABLE tabela2(
tabela_id NUMBER,
rep_vendas VARCHAR(15));
```

FIGURA 11 – BASE MERGE



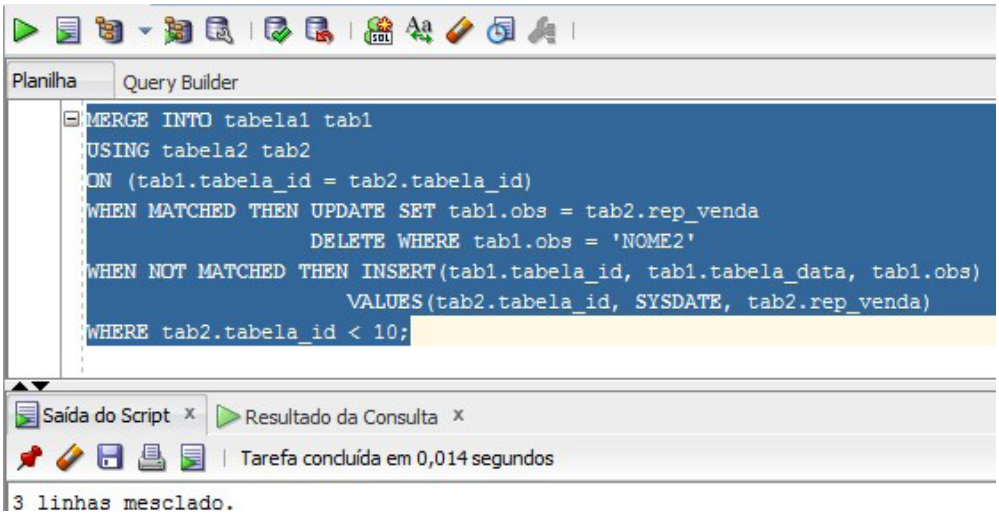
FONTE: Costa (2015, s.p.)

Para facilitar nosso entendimento, vejamos a definição dado por Costa (2015, s.p.):

> Fizemos um MERGE das linhas da tabela2 na tabela1, mas apenas as linhas da tabela2 que tivesse o valor em tabela_id menor que 10, ou seja apenas 3 linhas da tabela2 satisfazem essa condição. Quando existisse o mesmo id na tabela 1 ela seria atualizada, mas depois da atualização a coluna tabela_data fosse uma data de mais de dois anos atrás ela seria excluída, caso não seria inserida uma nova linha.

Como podemos perceber, esse comando pode ser muito útil, quando queremos alterar a base, pois podemos testar os resultados. Caso uma situação não seja atendida, ele fará a outra.

FIGURA 12 – MERGE



FONTE: Costa (2015, s.p.)

Veja mais sobre Merge em: https://bit.ly/2VtAVR1.

d. TCL (Transaction control language) Commit e Rollback

Após realizar uma alteração no banco, através de uma instrução do tipo DML, há a necessidade de confirmar ou descartar. O COMMIT confirma todas as alterações pendentes, realizadas no banco. Já o ROLLBACK desfaz todas alterações realizadas.

Sintaxe: **Commit; ou Rollback;**

Após executar um desses comandos, as alterações estarão disponíveis para todos os usuários que possuírem acesso à tabela.

e. DQL (Data Query Language) - linguagem de consulta de dados

A Data Query Language permite extrair dados do banco de dados através do comando SELECT. Semelhante ao UPDATE e DELETE sem WHERE, ao utilizarmos o comando SELECT podemos definir restrições (WHERE) e relacionamento com outras tabelas. A sintaxe tradicional SQL89, e que utilizaremos em nosso livro é:

```
SELECT [distinct] coluna1, coluna2, colunaN
from tabela1, tabela2, tabelaN
where condições
[group by colunas]
[having [condição]
[order by colunas]
```

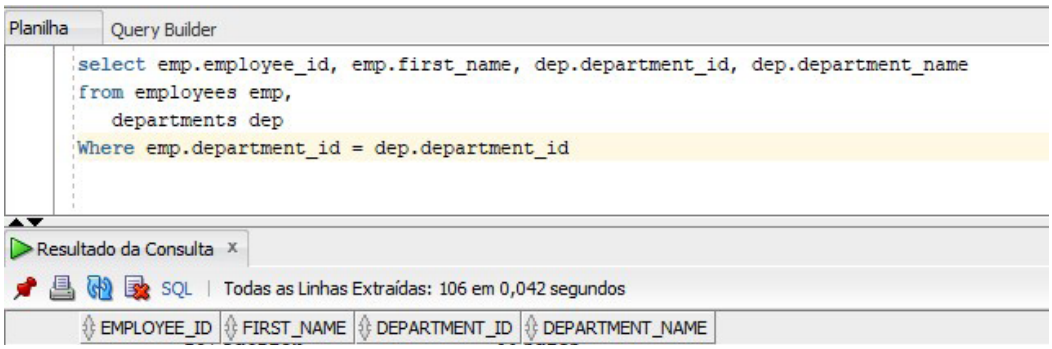
Entraremos em detalhe em cada uma das condições. Neste apresentaremos o funcionamento e os tipos de relacionamento existentes. As expressões entre [ ] são opcionais. Vamos conhecer os padrões existentes, começando pelo Inner join.

Em algumas consultas, utilizaremos a tabela. Ela é utilizada para fazer operações com SELECT onde não é necessário fazer extração de dados em tabelas. Normalmente quando queremos buscar a data, fazemos "SELECT sysdate from dual;". Ela basicamente ajuda a manter a sintaxe correta de um SELECT onde não teremos uma tabela na consulta, como, por exemplo, para realizar um cálculo "SELECT 2*18 from dual;".

f. INNER JOIN

O resultado desse tipo de Join será o mesmo do padrão tradicional, pois é utilizado quando o resultado esperado entre duas ou mais tabelas seja por coincidência, ou seja, para cada linha da primeira tabela teremos a(s) linha(s) da segunda tabela correspondente(s), segundo Alves e Júnior (2015). Vejamos o exemplo abaixo entre as tabelas EMPLOYEES e DEPARTMENTS.

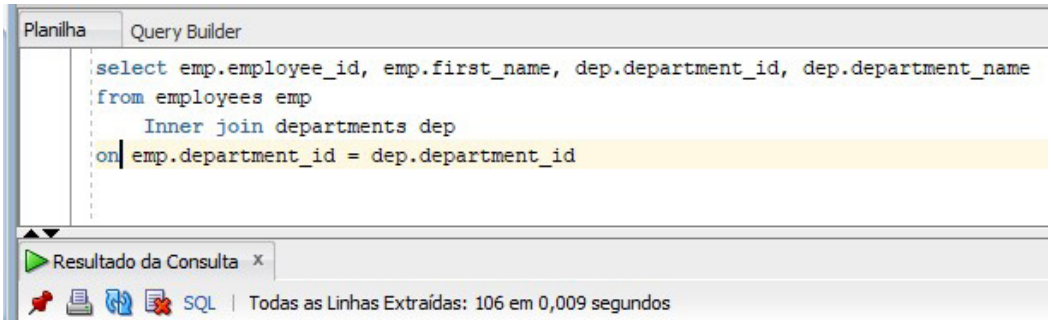
FIGURA 13 – PADRÃO TRADICIONAL (SQL89)



FONTE: O autor

Como podemos observar pela quantidade de linhas apresentadas nas imagens. Independendente do tipo de consulta realizada na mesma base e com o mesma restrição, elas retornam à mesma quantidade de linhas.

FIGURA 14 – PADRÃO ANSI92 (SQL92)

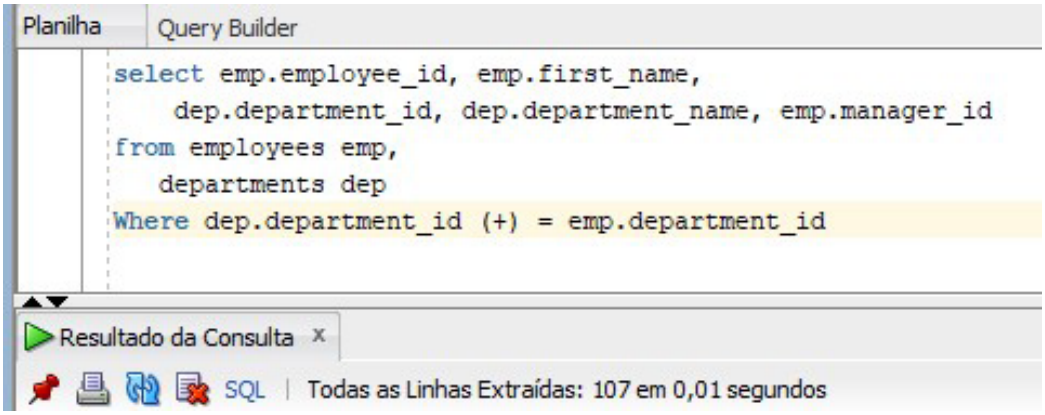


FONTE: O autor

g. LEFT JOIN

Caso uma das tabelas não tenha um valor correspondente, este tipo de Join deverá ser usado. Assim, irá retornar às linhas da primeira tabela mesmo que não haja correspondência na segunda tabela. Vejamos o padrão tradicional e o ANSI92. Perceba que o número de linhas retornado é o mesmo.

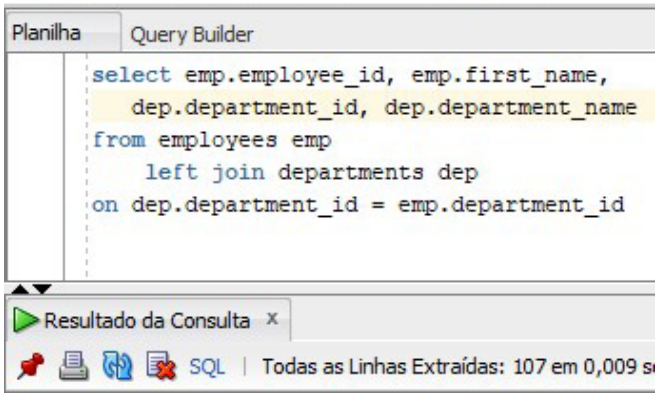
FIGURA 15 – PADRÃO TRADICIONAL (SQL89)



FONTE: O autor

Conforme observado anteriormente, o retorno será o mesmo. Você pode escolher entre o modelo utilizado.

FIGURA 16 – PADRÃO ANSI92 (SQL92)



FONTE: O autor

h. RIGHT JOIN

Semelhante ao LEFT JOIN, também não buscará por coincidência, só que agora na tabela direita. Ambas as consultas retornam o mesmo valor, a diferença no resultado pode ser vista, pois há departamentos que não têm funcionários alocados. Como vemos na figura a seguir:

FIGURA 17 – RIGHT JOING

107	(null)	(null)	120	Treasury
108	(null)	(null)	130	Corporate Tax
109	(null)	(null)	140	Control And C...
110	(null)	(null)	150	Shareholder S...
111	(null)	(null)	160	Benefits
112	(null)	(null)	170	Manufacturing
113	(null)	(null)	180	Construction
114	(null)	(null)	190	Contracting
115	(null)	(null)	200	Operations
116	(null)	(null)	210	IT Support
117	(null)	(null)	220	NOC
118	(null)	(null)	230	IT Helpdesk
119	(null)	(null)	240	Government Sales
120	(null)	(null)	250	Retail Sales
121	(null)	(null)	260	Recruiting
122	(null)	(null)	270	Payroll

FONTE: O autor

Vejamos duas soluções diferentes, uma utilizando o padrão SQL89 e outro utilizando SQL92. A escolha entre um ou outro, vai depender do SGBD utilizado e da sua preferência.

SQL89: SELECT emp.employee_id, emp.first_name, dep.department_id, dep.department_name
FROM employees emp, departments dep
WHERE dep.department_id = emp.department_id (+);
SQL92: SELECT emp.employee_id, emp.first_name,
dep.department_id, dep.department_name
FROM employees emp
right join departments dep
ON dep.department_id = emp.department_id;

i. FULL JOIN

Neste caso, segundo Alves e Júnior (2015), há a junção do INNER JOIN com a listagem de todas as outras linhas não associadas, tanto do lado direito RIGHT JOIN quanto do lado esquerdo LEFT JOIN.

FIGURA 18 – FULL JOING

- **Padrão ANSI92 (SQL92) – Exemplo 1:**

```
SELECT E.EMPLOYEE_ID,
       E.LAST_NAME,
       E.DEPARTMENT_ID,
       D.DEPARTMENT_ID,
       D.LOCATION_ID
  FROM EMPLOYEES E
       FULL JOIN DEPARTMENTS D
              ON D.DEPARTMENT_ID = E.DEPARTMENT_ID;
```

- **Padrão ANSI92 (SQL92) – Exemplo 2:**

```
SELECT E.EMPLOYEE_ID,
       E.LAST_NAME,
       E.DEPARTMENT_ID,
       D.DEPARTMENT_ID,
       D.LOCATION_ID
  FROM (SELECT *
          FROM EMPLOYEES
         WHERE EMPLOYEE_ID = 200
       ) E
       FULL JOIN DEPARTMENTS D
              ON D.DEPARTMENT_ID = E.DEPARTMENT_ID;
```

FONTE: Alves e Júnior (2015, s.p.)

É importante observar que NÃO sendo possível utilizar filtros nas tabelas (Exemplo 1), mas caso seja necessário, é preciso executar o SELECT antes na tabela colocando os filtros para depois executar o FULL JOIN (Exemplo 2). No modelo tradicional SQL89, não é possível executar esse tipo de comando.

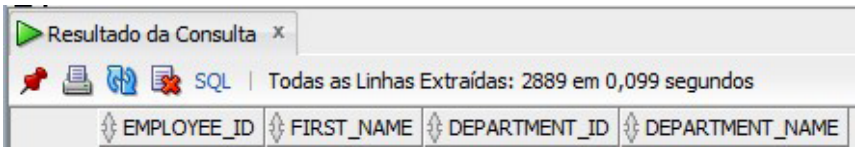
j. CROSS JOIN

Este último modelo deve apenas ser usado quando for extremamente necessário, pois ele junta duas ou mais tabelas por cruzamento. Ou seja, ele não faz validação alguma entre chaves, pois para cada linha da tabela EMPLOYEES queremos todos os DEPARTMENTS ou vice-versa. Ele também é chamado de produto cartesiano entre duas tabelas. Lembram o update ou delete sem WHERE, agora temos o SELECT sem WHERE.

SQL89	SQL92
SELECT emp.employee_id, emp.first_name, dep.department_id, dep.department_name from employees emp, departments dep	SELECT emp.employee_id, emp.first_name, dep.department_id, dep.department_name from employees emp cross join departments dep

O resultado de ambos os SELECT s é de 2889.

FIGURA 19 – CROSS JOIN



FONTE: O autor

O produto cartesiano é feito da seguinte maneira. A tabela Employees tem 107 registros, a tabela Departments, 27. Ela faz para cada registro da primeira tabela vezes os registros da segunda tabela. Dado dois conjuntos A e B, não vazios, denominamos produto cartesiano A x B o conjunto de todos os pares ordenados (x, y) onde

$$x \in A \quad e \quad y \in B$$

$$A \; X \; B = \{(x, y) \;\; x \in A \text{ e } y \in B \}$$

FONTE: <https://bit.ly/2y02qcE>. Acesso em: 13 mar. 2020.

k. Usando cláusula WHERE

Além de relacionar duas tabelas como vimos, podemos restringir os dados que buscamos, por exemplo, buscar os salários que sejam maiores que determinado valor, ou um nome que contenha determinada sequência de letras, ou que estejam em determinado conjunto de valores. Para isso podemos complementar a clausula WHERE, usando um ou mais restrições.

Sintaxe: **SELECT coluna1, coluna2, colunan**
FROM Tabela1
WHERE coluna operador condição
AND coluna operador condição;

QUADRO 16 – OPERADORES

OPERADOR	SIGNIFICADO	EXEMPLO
=	Igual a	SELECT salary, commission_pct from employees where salary = 24000 And commission_pct is not null
>	Maior que	SELECT salary from employees where salary > 24000
>=	Maior igual	SELECT salary from employees where salary >= 24000
<	Menor que	SELECT salary from employees where salary < 24000
<=	Menor idêntica	SELECT salary from employees where salary <= 24000
<>	Diferente	SELECT salary from employees where salary <> 24000
Between ...AND ...	Entre dois valores(inclusive)	SELECT salary from employees where salary BETWEEN 24000 and 45000
IN ( )	Algum valor do conjunto	SELECT salary from employees where salary in(24000,45000) **Só trará os que tiverem esses salários.
LIKE	Corresponde a um padrão de letras	SELECT First_name from employees where upper(first_name) like '%A' ** Somente os que tem a última letra 'A'; SELECT First_name from employees where upper(first_name) like 'A%'; ** Qualquer que começe com 'A'; SELECT First_name from employees where upper(first_name) like '%A%'; **Que tenha 'A' em qualquer posição;
IS NULL	É um valor nulo	SELECT salary from employees where salary is null
IS NOT NULL	É um valor não nulo	SELECT salary from employees where salary is not null

FONTE: O autor

i. Principais funções

Nesta seção, vamos conhecer as principais funções para utilizar em um banco de dados. Apresentaremos o modelo na linguagem Oracle. Existem funções de caractere, funções numéricas, de data e de grupos.

UPPER: A função UPPER transforma em maiúsculo os valores informados. Sintaxe: (coluna\expressão)

Exemplo: **SELECT upper('Hello World') from dual;**
SELECT UPPER(last_name) FROM employees;
LOWER: Transforma em minúsculo os valores informados. Sintaxe: Lower(coluna\expressão)

Exemplo: **SELECT lower('Hello World') from dual;**
SELECT lower(last_name) FROM employees;
INITCAP: Converte apenas a primeira letra em maiúsculo, todas as outras permanecem minúsculo. Sintaxe: initcap(coluna\expressão)

Exemplo**:** SELECT **initcap('hello world') from dual;**
SELECT initcap(last_name) FROM employees;

FIGURA 20 – UPPER, LOWER E INITCAP

```
select last_name original,
upper(last_name)Maius,
lower(last_name)Minusc,
Initcap(last_name)so_uma
from employees;
```

Resultado da Consulta

SQL | 50 linhas extraídas em 0,004 segundos

	ORIGINAL	MAIUS	MINUSC	SO_UMA
1	Abel	ABEL	abel	Abel
2	Ande	ANDE	ande	Ande
3	Atkinson	ATKINSON	atkinson	Atkinson
4	Austin	AUSTIN	austin	Austin
5	Baer	BAER	baer	Baer

FONTE: O autor

CONCAT: Concatena o primeiro valor com o segundo. Somente dois valores. O "||" pode concatenar diversos valores. Sintaxe: Concat(coluna1\expressão1, coluna2\expressão2)

Exemplo: **SELECT concat('hello', 'world') from dual;**
SELECT concat('nome ',last_name) FROM employees;

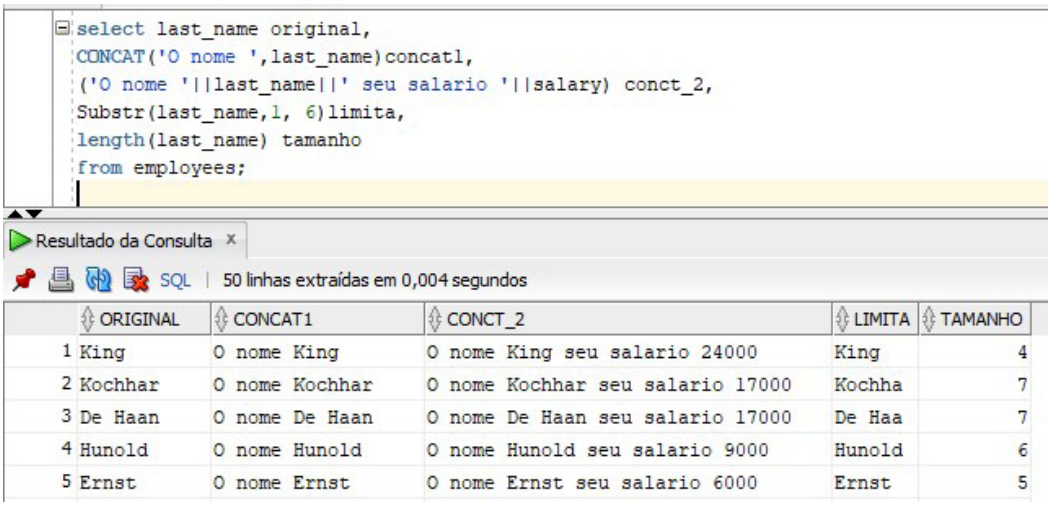
Existe outra maneira de concatenar várias colunas e expressões utilizando '||' (pipe) conforme a sintaxe**:** coluna1\expressão1|| coluna2\expressão2|| colunan\expressãon;

SUBSTR: A função retorna os caracteres especificados a partir do valores que inicia em m,n caracteres. Se o valor n for omitido, todos os caracteres até o final serão apresentados. Sintaxe: Substr(coluna\expressão, m[n]);

Exemplo: SELECT substr('hello world', 2, 5) from dual;
****neste exemplo, a partir da segunda posição ele contará 5 caracteres**
Exemplo: SELECT last_name, substr(last_name, 3, 3) FROM employees;
****neste exemplo, a partir da terceira posição ele contará 3 caracteres**
LENGTH: Retorna o número de caracteres da expressão ou coluna. Sintaxe: Length(coluna\expressão).

Exemplo: SELECT length('hello world') from dual;
Exemplo: SELECT last_namelength(last_name) FROM employees;

FIGURA 21 – CONCAT, SUBSTR, LENGTH



	ORIGINAL	CONCAT1	CONCT_2	LIMITA	TAMANHO
1	King	O nome King	O nome King seu salario 24000	King	4
2	Kochhar	O nome Kochhar	O nome Kochhar seu salario 17000	Kochha	7
3	De Haan	O nome De Haan	O nome De Haan seu salario 17000	De Haa	7
4	Hunold	O nome Hunold	O nome Hunold seu salario 9000	Hunold	6
5	Ernst	O nome Ernst	O nome Ernst seu salario 6000	Ernst	5

FONTE: O autor

INSTR: Retorna à posição onde a expressão foi encontrada. O valor default de m e n é 1, ou seja, começa a pesquisa desde o início até entrar a primeira ocorrência.

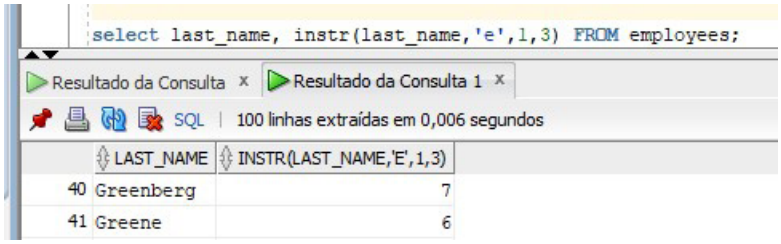
Sintaxe: INSTR(coluna\expressão, 'string',[m],[n])
Exemplo 1: **SELECT instr('hello world', 'lo') from dual;**
Exemplo 2: **SELECT last_name, instr(last_name,'e',1,3) FROM employees;**

No exemplo 1, o resultado será 4, como foram omitidos os parâmetros, ele buscará desde a primeira posição a primeira ocorrência.

No segundo exemplo, ele buscará desde a primeira letra, a terceira ocorrência da letra 'e'. Caso eu pesquise por 'E', não retornará linha alguma, pois ele difere maiúsculas de minúsculas. Para garantir que trará todos, o ideal é usar a função Upper ou lower aninhadas. Quando utilizamos uma função dentro de outra, chamam-se funções aninhadas.

SELECT last_name, instr(upper(last_name),'E',1,3) FROM employees;
SELECT last_name, instr(lower(last_name),'e',1,3) FROM employees;

FIGURA 22 – INSTR

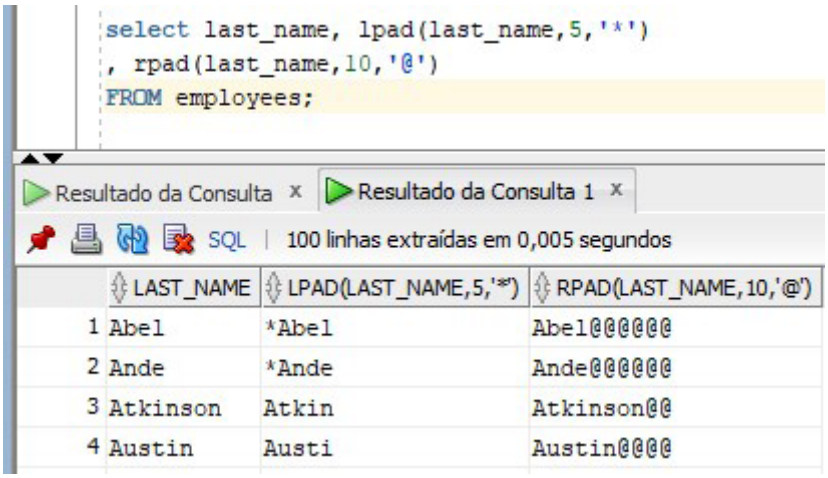


FONTE: O autor

LPAD e RPAD: As iniciais L e R, correspondem a iniciais de esquerda (**L**eft) e direita (**R**ight). A função é a mesma, só que dependendo da inicial, se aplicará na direita ou na esquerda. A função insere caracteres com a largura de n caracteres. Ou seja, o caracter que definirmos será incluído até atingir o limite informado.

Sintaxe:Lpad(coluna\expressão, n, 'string') ou Rpad(coluna\expressão, n, 'string')

FIGURA 23 – LPAD E RPAD



FONTE: O autor

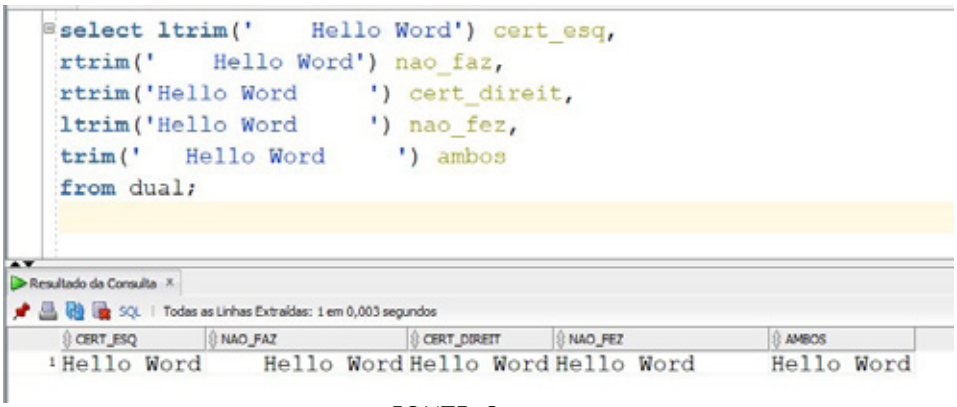
Observe no resultado da execução, na primeira coluna temos o valor armazenado no banco, na segunda coluna estamos aplicando a função LPAD, ela irá preencher com o símbolo '*' até atingir 5 caracteres, e a função RPAD irá preencher com @ até atingir 10 caracteres. Como o nome 'ABEL' já tem 4 caracteres, ela só acrescentará um '*' à esquerda, e na última coluna, é preenchido até o limite de 10 caracteres.

TRIM, LTRIM e RTRIM: A função **retira caracteres com a largura de n caracteres. Ou seja, o caracter que eu definir será excluído o até atingir o ca**racter informado.

Sintaxe: **TRIM (coluna\expressão, [trim_string])**
RTRIM (coluna\expressão, [trim_string])
LTRIM (coluna\expressão, [trim_string])

Na imagem a seguir, nós aplicamos os três modelos, na linha um os espaços estão do lado esquerdo, e como o comando é LTRIM, ele removeu os espaços. A mesma frase, aplicando o comando RTRIM não acontece nada, foi o que fizemos nas linhas três e quatro. Na última linha de instrução, aplicamos apenas o TRIM, ele remove os espaços de ambos os lados.

FIGURA 24 – TRIM



FONTE: O autor

REPLACE: A função REPLACE procura uma palavra ou conjunto de caracteres em uma coluna ou em uma expressão e substitui por outra. No exemplo apresentado, vai substituir a

Sintaxe: REPLACE(coluna\expressão, palavra procurada, palavra substituta)
Exemplo: **SELECT REPLACE('Hello word','word','world') from dual;**

3 FUNÇÕES COM NÚMEROS

Agora que já conhecemos as funções de textos, vamos conhecer as funções de número, são elas ROUND, TRUNC e MOD.

Vamos observar a figura a seguir, a função ROUND irá arredondar os valores, pode-se informar o número de casa decimais. A função TRUNC irá truncá-los, e a quantidade de casas decimais. Já a função MOD irá retornar o resultado de uma divisão.

Sintaxe: **round(coluna\expressão, n)**
trunc(coluna\expressão, n)
mod(m,n)

FIGURA 25 – FUNÇÕES NUMÉRICAS

```
select round(56.926), round(56.926,2),
       trunc(56.926),trunc(56.926,2),
       mod(1600,300), mod(2100,300) from dual;
```

ltado da Consulta x

SQL | Todas as Linhas Extraídas: 1 em 0,002 segundos

ROUND(56.926)	ROUND(56.926,2)	TRUNC(56.926)	TRUNC(56.926,2)	MOD(1600,300)	MOD(2100,300)
57	56,93	56	56,92	100	0

FONTE: O autor

4 FUNÇÕES COM DATAS

Em algumas situações, há necessidade de trabalhar com datas, para isso algumas funções podem auxiliar. Na proxima unidade vamos conhecer um pouco mais sobre datas e como aplicar máscaras na sua formatação.

QUADRO 17 – FUNÇÕES COM DATA

Função	Descrição
MONTS_ BEETWEEN	Retorna o número de meses entre duas datas. MONTS_ BEETWEEN(data1, data2).
ADD_ MONTHS	Adiciona meses do calendário à data. ADD_MONTHS(data, n)
NEXT_DAY	Dia seguinte ao da data especificada. NEXT_DAY(date, 'dia')
LAST_DAY	Último dia do mês. LAST_DAY(data)
ROUND	Arredonda a data. ROUND(data, 'formato') Se o formato de data não for informado, a data será arrendanda até o dia mais próximo.
TRUNC	Trunca a data. TRUNC (data, 'formato') Se o formato de data não for informado, a data será truncada até o dia mais próximo.

FONTE: O autor

Vejamos, na prática, algumas utilizações das funções de data. No exemplo apresentado a seguir, estamos buscando a quantidade de meses entre duas datas. Na segunda linha, estamos adicionando 120 meses à data de 25/08/2015. As demais funções já foram detalhadas acima.

FIGURA 27 – FUNÇÕES DE DATA

```
select months_between('22/09/2004', '14/10/215')meses,
add_months('25/08/2015', 120)soma,
next_day('01/01/2010','segunda')prox,
LAST_DAY('18/10/2019')ult_dia,
round(sysdate,'MM')mes_ro,
round(sysdate,'YY')ano_ro,
trunc(sysdate,'MM')mes_tr,
trunc(sysdate,'YY')ano_tr
from dual;
```

ltado da Consulta ×

SQL | Todas as Linhas Extraídas: 1 em 0,003 segundos

MESES	SOMA	PROX	ULT_DIA	MES_RO	ANO_RO	MES_TR	ANO_TR
21467,2...	25/08/25	04/01/10	31/10/19	01/11/19	01/01/20	01/10/19	01/01/19

FONTE: O autor

Vejamos algumas funções que são úteis, chamadas de funções gerais.

5 FUNÇÕES GERAIS

São funções que podem ser usadas tanto com números, quanto com textos e datas. A função NVL, converte um valor nulo em um valor válido. Vejamos como aplicar esse comando.

Sintaxe: NVL(valor, retorno).

Exemplo: SELECT NVL(commission_pct,0) from employees e neste exemplo, se o empregado não tiver comissão, será atribuído o valor zero, e se ele tiver valor, será apresentado o valor da comissão. Vejamos um exemplo utilizando uma coluna texto (varchar2).

Exemplo: SELECT nvl(state_province,'Sem_nome') from locations;

LEITURA COMPLEMENTAR

NORMALIZAÇÃO

Diego Machado

Normalização é o processo de modelar o banco de dados projetando a forma como as informações serão armazenadas a fim de eliminar, ou pelo menos minimizar, a redundância no banco. Tal procedimento é feito a partir da identificação de uma anomalia em uma relação, decompondo-as em relações mais bem estruturadas.

Normalmente precisamos remover uma ou mais colunas da tabela, dependendo da anomalia identificada e criar uma segunda tabela, obviamente com suas próprias chaves primárias e relacionarmos a primeira com a segunda para assim tentarmos evitar a redundância de informações.

> O processo de normalização compreende o uso de um conjunto de regras, chamados de formas normais. Ao analisarmos o banco de dados e verificarmos que ele respeita as regras da primeira forma normal, então podemos dizer que o banco está na "primeira forma normal". Caso o banco respeite as primeiras três regras, então ele está na "terceira forma normal". Mesmo existindo mais conjuntos de regras para outros níveis de normalização, a terceira forma normal é considerada o nível mínimo necessário para grande parte das aplicações (MICROSOFT 2007).

Um banco de dados dentro dos padrões de normalização reduz o trabalho de manutenção e ajuda a evitar o desperdício do espaço de armazenamento. Se tivermos cadastrado no banco um cliente e tivermos o seu telefone registrado em mais de uma tabela, havendo uma alteração no seu número de telefone, teremos que fazer essa atualização em cada tabela. A tarefa se torna muito mais eficiente se tivermos seu telefone registrado em apenas uma tabela.

Os próximos parágrafos demonstram melhor as anomalias no banco de dados e as diferentes regras de normalização, bem como a forma de aplicá-las para estruturarmos o banco de dados da melhor maneira possível.

Formas Normais

Como mencionado anteriormente, temos conjuntos de regras para determinar com qual forma normal o banco é compatível. Primeiramente, precisamos verificar se encontramos compatibilidade com a primeira forma normal. Caso esteja tudo conforme, analisamos se a segunda forma normal se encaixa e assim sucessivamente.

É importante lembrar que para uma relação atender as exigências de uma forma normal, se faz necessário que esta obedeça às regras da forma normal anterior. A primeira forma normal é exceção pois não existe uma forma normal anterior a primeira.

Primeira Forma Normal

Uma relação está na primeira forma normal quando todos os atributos contêm apenas um valor correspondente, singular e não existem grupos de atributos repetidos — ou seja, não admite repetições ou campos que tenham mais que um valor.

O procedimento inicial é identificar a chave primária da tabela. Após, devemos reconhecer o grupo repetitivo e removê-lo da entidade. Em seguida, criamos uma nova tabela com a chave primária da tabela anterior e o grupo repetitivo.

Código	Nome	Endereço	Telefone
1001	Diego Machado	Rua Tal 321 Porto	5312345678 5398765432
1002	Fulano de Tal	Avenida Tal 71 Centro	5187654321 5143215678

Tabela 1: Tabela não está na primeira forma normal

Analisando o exemplo acima, podemos observar dois problemas: temos uma pessoa com dois números de telefone e um endereço com diferentes valores, a rua e o bairro. A fim de normalizar, teremos que colocar cada informação em uma coluna diferente e criar uma nova tabela relacionando a pessoa a seus números de contato.

Código	Nome	Endereço	Bairro
1001	Diego Machado	Rua Tal 321	Porto
1002	Fulano de Tal	Avenida Tal 71	Centro

Tabela 2: Tabela está na primeira forma normal

Dessa forma, como mostrado na tabela acima, temos uma tabela na primeira forma normal evitando assim repetições e campos com múltiplos valores, conforme observamos na tabela abaixo.

Código	Telefone
1001	5312345678
1001	5398765432
1002	5112345678
1002	5187654321

Tabela 3: Nova tabela criada para evitar campos com mais de um valor

Segunda Forma Normal

É dito que uma tabela está na segunda forma normal se ela atende a todos os requisitos da primeira forma normal e se os registros na tabela, que não são

chaves, dependam da chave primária em sua totalidade e não apenas parte dela. A segunda forma normal trabalha com essas irregularidades e previne que haja redundância no banco de dados.

Para isso, devemos localizar os valores que dependem parcialmente da chave primária e criar tabelas separadas para conjuntos de valores que se aplicam a vários registros e relacionar estas tabelas com uma chave estrangeira.

cd_locacao	cd_filme	titulo_filme	devolucao	cd_cliente
1010	201	The Matrix	2011-10-12	743
1011	302	O Grito	2011-12-10	549
1012	201	The Matrix	2011-12-30	362

Tabela 4: Tabela não está na segunda forma normal

Podemos observar que a tabela acima apresenta uma coluna responsável por armazenar o título do filme, onde este foi alugado e está associado a um número de locação. Porém, ele também está associado a um código, tornando-o então um valor que não é totalmente dependente da chave primária da tabela.

cd_filme	titulo_filme
201	The Matrix
302	O Grito

Tabela 5: Tabela criada para armazenar os filmes

Se em algum momento tivermos que alterar o título de um filme, teríamos que procurar e alterar os valores em cada tupla (linha) da tabela. Isso demandaria um trabalho e tempo desnecessário. Porém, ao criarmos uma tabela e vincularmos elas com o recurso da chave estrangeira, tornamos o nosso banco mais organizado e ágil para as futuras consultas e manutenções que podem vir a ser necessárias.

cd_locacao	cd_filme	devolucao	cd_cliente
1010	201	2011-10-12	743
1011	302	2011-12-10	549
1012	201	2011-12-30	362

Tabela 6: Tabela na segunda forma normal

Terceira Forma Normal

Se analisarmos uma tupla e não encontrarmos um atributo **não chave** dependente de outro atributo **não chave**, podemos dizer que a entidade em questão

está na terceira forma normal – contanto que esta não vá de encontro às especificações da primeira e da segunda forma normal.

Como procedimento principal para configurar uma entidade que atenda as regras da terceira forma normal, nós identificamos os campos que não dependem da chave primária e dependem de um outro campo não chave. Após, separamos eles para criar uma outra tabela distinta, se necessário.

placa	modelo	qtd_kmetro	cod_fab	nome_fab
qwe1234	Modelo1	867	3004	fabricante1
asd456	Modelo2	928	3005	fabricante2

Tabela 7: Tabela não está na terceira forma normal

No exemplo acima temos uma entidade que lista os carros cadastrados, bem como o modelo, a quantidade de quilômetros rodados, o código do fabricante e o nome do fabricante. Observamos que "nome_fab" se dá em função de "cod_fab". Para adequarmos esta tabela de acordo com os padrões da terceira forma normal, devemos remover a coluna do nome do fabricante.

placa	modelo	qtd_kmetro	cod_fab
qwe1234	Modelo1	867	3004
asd456	Modelo2	928	3005

Tabela 8: Tabela na terceira forma normal

A coluna que removemos, deve ser colocada em uma nova tabela, relacionando corretamente o nome do fabricante com o seu código. Abaixo, podemos observar como ficaria esta nova entidade.

cod_fab	nome_fab
3004	fabricante1
3005	fabricante2

Tabela 9: Tabela criada para armazenar o nome do fabricante

Quarta Forma Normal

Uma entidade está na quarta forma normal quando ela estiver na terceira forma normal e não existir dependências multivaloradas entre seus atributos, ou seja, campos que se repetem em relação a chave primária, gerando redundância nas tuplas da entidade. Devemos fragmentar essa relação com o objetivo de não termos mais essas dependências funcionais do gênero.

"Em outras palavras, quando houver repetição de dois ou mais atributos não chave, gerando uma redundância desnecessária na tabela, dividimos essa tabela em dois ou mais subgrupos evitando assim o problema da redundância" (Silva 2010).

musica	artista	album
Música 1	Artista 1	Álbum 1
Música 1	Artista 2	Álbum 2
Música 1	Artista 1	Álbum 2
Música 2	Artista 3	Álbum 1
Música 2	Artista 2	Álbum 1
Música 2	Artista 3	Álbum 2

Tabela 10: Tabela não está na quarta forma normal

Conforme o exemplo acima, temos uma tabela relacionando música, cantor e álbum, contendo as músicas. Uma música pode ser interpretada por um artista e esta pode estar em um ou mais álbuns ou ser interpretada por outro artista. Para evitarmos a repetição de informações, devemos dividir a tabela.

musica	artista
Música 1	Artista 1
Música 1	Artista 2
Música 2	Artista 3
Música 2	Artista 2

Tabela 11: Tabela na quarta forma normal

musica	album
Música 1	Álbum 1
Música 1	Álbum 2
Música 2	Álbum 1
Música 2	Álbum 2

Tabela 12: Tabela na quarta forma normal

Conclusão

Considerando as dificuldades de elaborar um projeto para um sistema e planejar toda a modelagem de um banco de dados robusto, ágil e seguro, as regras para normalização de dados aplicadas da forma correta contribuem consideravelmente para a criação de uma boa estrutura das bases de dados relacionais, evitando anomalias de redundância ou perda de informação.

Dessa forma, a pessoa que vai analisar a documentação de uma modelagem normalizada consegue abstrair com mais clareza, pois uma vez conhecendo os padrões, a compreensão é facilitada e agiliza todo o trabalho. Como desvantagem podemos citar o aumento do número de tabelas. Bancos devidamente construídos, ou seja, na terceira forma normal, apresentam um número maior de tabelas em comparação aos bancos não normalizados. Assim, as consultas em bancos com mais tabelas requerem uma complexidade maior na elaboração do SQL, fazendo necessário o uso dos *Joins* e cláusulas para elaborarmos a consulta adequadamente.

Dado o exposto, a aplicação das regras de normalização de dados é altamente recomendada, pois os ganhos são consideravelmente relevantes. Investir um pouco mais de dedicação e tempo trabalhando com um número maior de tabelas traz mais benefícios do que um banco de dados sem a devida organização.

FONTE: <https://medium.com/@diegobmachado/normaliza%C3%A7%C3%A3o-em-banco-de-dados-5647cdf84a12>. Acesso em: 22 mar. 2020.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- Para dar manutenção nos dados da base, usamos o comando insert, através da seguinte sintaxe: INSERT INTO table [(column[,colum..])] VALUES (value[, value]);
- A inserção pode ser feita de duas maneiras, uma indicando as colunas e a outra seguindo a ordem de criação.
- Para realizarmos a manutenção dos dados, usamos o comando update, através da seguinte sintaxe: UPDATE table SET column = value [, column = value, …] [where condition];
- É de suma importância a utilização do comando Where na limitação dos dados a serem alterados.
- Para apagarmos os dados, utilizamos o comando delete com a seguinte Sintaxe: DELETE [FROM] table [WHERE condition].
- O comando Truncate remove todas a linhas, mas mantém a estrutura da tabela na base.
- Para confirmar uma transação usamos Commit e para desfazer utilizamos o comando Rollback.
- O comando DQL (Data query languagem), permite fazer seleções na base de dados.

Ficou alguma dúvida? Construímos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

AUTOATIVIDADE

Com base no modelo criado no tópico anterior, vamos aplicar os comando DML de INSERT, UPDATE E DELETE.

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

1 Inserir os dados nas tabelas, conforme o schema apresentado. Não se esqueça de commitar os dados na base.

2 Com base nos conhecimentos da álgebra relacional e nos comandos de seleção. Vamos recuperar o número do funcionário e o seu nome e sua data de admissão, somente os que forem do sexo 'M'.

3 Altere a coluna sexo, do funcionário 'Pedro Sergio' para 'O' (indicando outro). Não esqueça de utilizar a chave primária como forma de restrição.

UNIDADE 3

BANCO DE DADOS FUNÇÕES E PROGRAMAÇÃO

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você deverá ser capaz de:

- conhecer e manipular a apresentação dos dados através de funções;
- entender de que forma os objetos do Banco de Dados interagem entre si para a resolução de problemas;
- criar algoritmos e fazer manutenções em objetos do Banco de Dados;
- construir rotinas de programação avançada em Banco de Dados, através de Blocos anônimos, exceptions, procedures e funções.

PLANO DE ESTUDOS

Esta unidade está dividida em três tópicos. No decorrer da unidade, você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – MÁSCARAS E OUTRAS FUNÇÕES

TÓPICO 2 – PL/SQL, BLOCOS ANÔNIMOS E EXCEÇÕES

TÓPICO 3 – PROCEDURES E FUNCTIONS

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

TÓPICO 1

MÁSCARAS E OUTRAS FUNÇÕES

1 INTRODUÇÃO

Durante nosso livro didático, vimos toda a estrutura de um banco de dados relacional, desde a sua origem até a forma de como criar uma base de dados.

Muitas vezes, a forma como os dados foram armazenados no banco, precisam ser trabalhados para melhor entendimento do usuário. Para isso, neste tópico, vamos conhecer algumas funções e máscara ou *alias*.

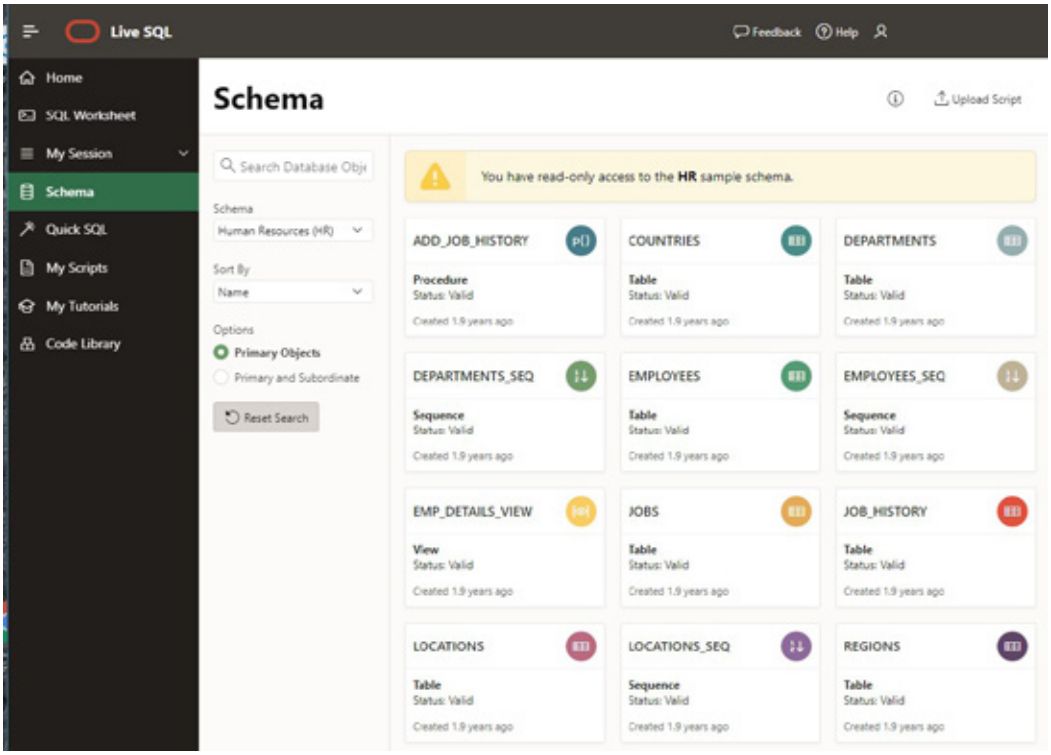
Além disso, podemos implementar funções e procedimentos, que auxiliaram na obtenção de rotinas que não precisam ser controladas pela aplicação. Quanto mais conhecimentos tivermos como desenvolver essas funcionalidades, melhor será o aproveitamento do nosso banco de dados relacional.

2 MÁSCARAS

Dando continuidade aos comandos vistos na Unidade 2, vamos ver mais algumas funções de conversão, que serão muito úteis na extração e manipulação de dados. Na Unidade 2 do nosso livro didático, já vimos algumas como o LOWER, UPPER, que servem para transformar valores em minúsculos e maiúsculos respectivamente. Vimos também algumas funções que utilizam datas. Agora vamos ver como utilizar máscara em datas e números.

Caso você não queira instalar o banco de dados na sua máquina, você pode criar gratuitamente uma conta na página da Oracle e fazer os exercícios, como utilizaremos aqui. O endereço é: https://rextester.com/l/oracle_online_compiler.
Existem outras ferramentas on-line para a execução de comando, como o https://rextester.com/l/oracle_online_compiler.

FIGURA 1 – CONEXÃO LIVE SQL



FONTE: O autor

Após a criação da sua conta, vamos utilizar o mesmo *schema* para que você possa reproduzir os exercícios executados. Como vemos na Figura 1, você deve escolher o *schema* Human Resources HR. A tela onde executaremos os comandos é apresentada na Figura 2.

FIGURA 2 – EXECUTANDO

FONTE: O autor

Para a execução da instrução, basta escrever o código e clicar em Run, que o resultado será apresentado, conforme apresentado na Figura 2. Agora que já conhecemos a ferramenta, vamos começar com as funções de conversão. Segundo Fanderuff (2003), essas funções servem para a conversão de datas, números e textos, sendo muito utilizadas para a formatação da apresentação de datas e números. O formato para a conversão é:

SINTAXE: **TO_CHAR(data [coluna, formato_char]) ou TO_CHAR (número[coluna, formato_char])**

Vejamos as possíveis formas de conversão, propostas por Fanderuff (2003).

FIGURA 3 – CONVERSÃO

Data

Formatos válidos:
CC — Século
SCC — Século precedido de "-" quando for (BC)
YYYY — Ano com 4 posições
SYYYY — Ano com 4 posições precedido de "-" quando for (BC)
YYY ou YY ou Y — Ano com 3, 2 ou 1 posição
YEAR — Ano por extenso
SYEAR — Ano por extenso precedido de "-" quando for (BC)
BC ou AD — Indicador de antes de Cristo (BC) ou depois de Cristo (AD)
B.C. ou A.D. — Idem anterior
Q — Trimestre
MM — Mês em representação numérica (01-12)
MONTH — Mês por extenso
MON — Abreviação com as 3 primeiras letras do mês
RM — Mês em algarismos romanos
WW — Semana do ano (01-52)
W — Semana do mês (1-5)
DDD — Dia do ano (1-366)
DD — Dia do mês (1-31)
D — Dia da semana (1-7)
DAY — Dia da semana por extenso
DY — Abreviação com 3 letras do dia da semana
J — Data no formato juliano (número de dias a partir de 1/01/4712 BC)
AM ou PM — Antes do meio-dia (AM) e depois do meio-dia (PM)
A.M. ou P.M. — Idem ao anterior
HH ou HH12 — Hora do dia (1-12)
HH24 — Hora do dia (1-24)
MI — Minutos (1-59)
SS — Segundos (1-59)
SSSSS — Segundos após a meia-noite (0-86399)

Prefixo:
FM — Retirar brancos à esquerda e zeros à direita

Sufixos:
TH — Número ordinal
SP — Número por extenso

Numeros

Formatos válidos:
9 — Normal
. — Para casas decimais (ponto)
G — Para casas decimais
, — Separador de algarismos (vírgula/milhar)
D — Separador de algarismos (milhar)

Prefixos:
0 — Com zeros no início
$ — Com o caractere $ no início
B — Valor zerado é exibido em branco

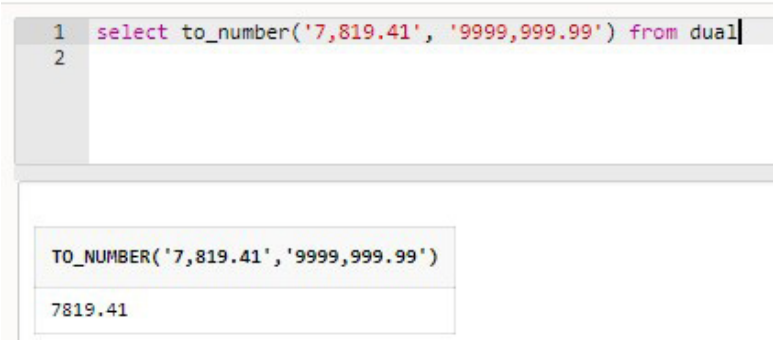
Sufixos:
MI — O "-" de números negativos aparecerá no final do número
PR — Números negativos aparecem no formato "<n>"

FONTE: Adaptado de Fanderuff (2003, p. 134)

A formatação da data no banco, vai depender da configuração do banco. Nos nossos exemplos, segue o padrão americano de mês/dia/ano.

Como apresentado na Figura 2, onde convertemos uma data para um texto, podemos fazer o mesmo com números. Vejamos, o exemplo onde convertemos um texto para um número.

FIGURA 4 – FORMATANDO NÚMERO



FONTE: O autor

Existem diversos artigos e foruns na internet para a instalação e criação de base de dados. Acesse: https://bit.ly/39REaqo e https://bit.ly/2UWJKUc.

Vejamos outros exemplos de aplicação de aplicação de máscaras em campos númericos, segundo Price (2009).

QUADRO 1 – FUNÇÃO TO_CHAR

Função TO_CHAR	**Resultado**
TO_CHAR(12345.67, '99999.99'),	12345.67
TO_CHAR(12345.67, '99,999.99') ,	12,345.67
TO_CHAR(12345.67, '099,999.99') ,	012,345.67
TO_CHAR(12345.67, '99,999.9900'),	12,345.6700
TO_CHAR(12345.67, '$99,999.99') ,	$12,345.67
TO_CHAR(0.67, 'B9.99') ,	.67
okTO_CHAR(12345.67, 'C99,999.99'),	BRL12,345.67
TO_CHAR(12345.67, '99999D99'),	12345,67
TO_CHAR(12345.67,'99999.99EEEE'),	1.23E+04
TO_CHAR(0012345.6700,'FM99999.99'),	12345.67
TO_CHAR(12345.67, '99999G99'),	123.46
TO_CHAR(12345.67, 'L99,999.99'),	R$12,345.67
TO_CHAR(-12345.67, '99,999.99MI'),	12,345.67-
TO_CHAR(-12345.67, '99,999.99PR'),	<12,345.67>
TO_CHAR(2007, 'RN') ,	MMVII
TO_CHAR(12345.67, 'TM'),	Cr$12,345.67
TO_CHAR(12345.67, 'U99,999.99'),	1234567
TO_CHAR(12345.67, '99999V99'),	12345,67
TO_CHAR(12345.67,'FM999G990D00'),	12.345,67
TO_CHAR (12345.67,'FM999990D00')	12345,67

FONTE: Adaptado de Price (2009)

Essa formatação de idioma, pode ser alterada via comando SQL, ou deve-se solicitar ao DBA (Administrador do banco de dados) a mudança, caso seja realmente necessário, pois o impacto é minimo. Veja o exemplo de alteração via comando SQL:

QUADRO 2 – ALTERAÇÃO VIA COMANDO SQL

Comando	Resultado
SELECT to_char(SYSDATE,('fmDAY, dd "de" MONTH "de" YYYY'),'nls_date_language = PORTUGUESE') AS DATA_LITERAL	DOMINGO, 5 de JANEIRO de 2020
,to_char(SYSDATE,('fmDAY, dd"," MONTH"," YYYY'), 'nls_date_language = AMERICAN') AS DATA_LITERAL_AMERICANA	SUNDAY, 5, JANUARY, 2020
,to_char(SYSDATE,'fmyyyy-MON-dd, DAY','nls_ date_language = AMERICAN') AS DATA_ PADRAO_AMERICANO	2020-JAN-5, SUNDAY
,to_char(SYSDATE,'fmDAY, dd/mm/yyyy','nls_ date_language = PORTUGUESE') AS DATA_ PADRAO	DOMINGO, 5/1/2020
,to_char SYSDATE, 'DAY', 'nls_date_language = PORTUGUESE') AS DIA_LITERAL	DOMINGO
,to_char (SYSDATE, 'MONTH','nls_date_language = PORTUGUESE') AS MES_LITERAL	JANEIRO
,to_char SYSDATE, 'DD', 'nls_date_language = PORTUGUESE') AS DIA	5
,to_char (SYSDATE, 'MM', 'nls_date_language = PORTUGUESE') AS MES	1
,to_char (SYSDATE, 'YYYY', 'nls_date_language = PORTUGUESE') AS ANO	2020
,to_char (SYSDATE, 'WW') AS SEMANA1	1
,to_char (SYSDATE, 'W') AS SEMANA2	1
,to_char (SYSDATE, 'IW') AS SEMANA3	1
,to_char (SYSDATE, 'HH24:MI:SS') AS HORA_ INTEIRA	10:28:55
,to_char (SYSDATE, 'HH24') AS HORA_24	10
,to_char (SYSDATE, 'HH') AS HORA_12	10
,to_char (SYSDATE, 'AM') AS "AM/PM"	MANHÃ
FROM dual;	

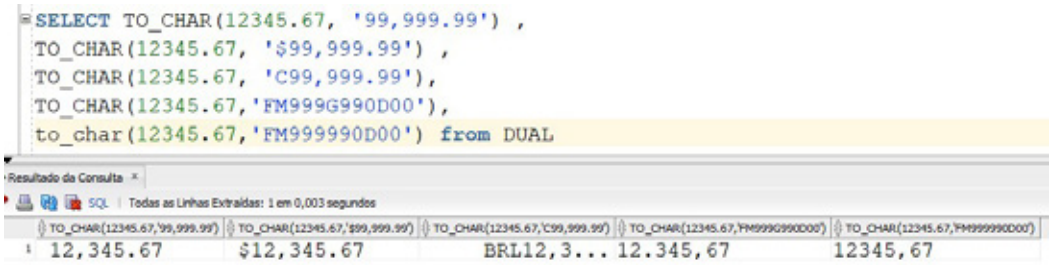
FONTE: O autor

Vale ressaltar, que ao executarmos esse tipo de consulta, seria difícil identificar cada uma das colunas. Para isso, utilizamos os apelidos ou alias.

3 APELIDOS

Os apelidos ou alias podem ser utilizados para colunas e para tabelas, os alias de coluna, são muito úteis em relatórios e na visualização das informações. Já os de coluna, evitam erros de identificação de colunas. Conforme podemos observar, quando utilizamos uma função, o resultado que aparece na execução, não é somente o nome da coluna, mas toda a instrução (conforme indicado pela seta). Para solucionar esse problema, podemos utilizar apelidos ou alias, segundo Fanderuff (2003).

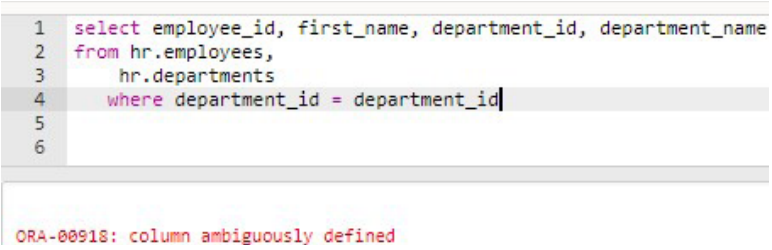
FIGURA 5 – ALIAS DE COLUNA



FONTE: O autor

Os alias de tabela, são necessários quando duas tabelas possuem a mesma coluna, para identificar de qual tabela se refere a coluna. Vejamos um exemplo, onde vamos buscar o nome do departamento e o nome do funcionário. Vejamos o exemplo desenvolvido com base no modelo HR.

FIGURA 6 – ALIAS DE TABELA



FONTE: O autor

Como podemos perceber, a ferramenta não consegue identificar qual tabela está sendo comparada na condição WHERE, por isso necessitamos definir um alias de tabela, conforme apresentado na Figura 6.

FIGURA 7 – ALIAS DE TABELA (2)

```
1  select emp.employee_id, emp.first_name,  dep.department_id, dep.department_name
2  from hr.employees emp,
3      hr.departments dep
4     where dep.department_id = emp.department_id
5
6
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Jennifer	10	Administration
201	Michael	20	Marketing
202	Pat	20	Marketing
114	Den	30	Purchasing
115	Alexander	30	Purchasing
116	Shelli	30	Purchasing
117	Sigal	30	Purchasing
118	Guy	30	Purchasing

FONTE: O autor

No exemplo apresentado na Figura 7, temos as possibilidade de utilização de alias de coluna, ele será utilizado na linha de seleção e ordenação. Na seleção ele e usado após a coluna que queremos utilizar o apelido, já na ordenação, eu utilizo apenas o apelido. Lembrando que se o apelido conter espaços em branco ou caracteres especiais (como # ou $) envolva o apelido com aspas duplas (“”). Podemos utilizar também “AS” para indicar o apelido, mas ele não é obrigatório. Na figura a seguir, temos um exemplo de como fica a execução no LiveSQL e no SqlDeveloper, respectivamente.

O *alias* de coluna e de tabela, não são excludentes, mas sim complementares, ou seja, você DEVE usar o alias de tabela para evitar erros de idenficação de qual tabela a coluna pertece. E o alias de coluna para melhorar a apresentação do resultado da coluna.

FIGURA 8 – APELIDO LIVESQL

```
1  select emp.employee_id "código do empregado",
2         emp.first_name as nome,
3         to_char(emp.hire_date, 'Day month year') dia
4  from hr.employees emp,
5     hr.departments dep
6     where dep.department_id = emp.department_id
7     order by nome, "código do empregado"
```

código do empregado	NOME	DIA		
121	Adam	Sunday	april	two thousand five
196	Alana	Monday	april	two thousand six
147	Alberto	Thursday	march	two thousand five
103	Alexander	Tuesday	january	two thousand six
115	Alexander	Sunday	may	two thousand three

FONTE: O autor

Na Figura 8, a data aparece em inglês, pois o banco está configurado dessa forma, como vimos anteriormente, pode ser configurado no banco ou via comando. No proximo tópico, iremos conhecer outras formas de trabalharmos a apresentação dos dados. No próximo item, conhecemos o DECODE e CASE.

4 DECODE E CASE

A utilização de funções auxilia muito a apresentação dos resultados, mas não são as únicas formas de melhorar a nossa seleção, e apresentar os resultados conforme nossa necessidade. As funções Case e Decode, também auxiliam na apresentação, vamos conhecê-las.

4.1 DECODE

A função CASE, segundo Price (2009), veio substituir a função DECODE, que era uma função utilizada até a versão 9i, ambas funcionam sendo apenas mais uma opção, vamos ver as duas formas. A Função DECODE funciona da seguinte forma, vejamos tomando como base a tabela DEPARTMENTS, do nosso modelo de dados HR:

FIGURA 9 – TABELA DEPARTAMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400

FONTE: O autor

Lembrando que a base permanece inalterada, para alterar os valores na base é necessário realizar o comando update e commit.

O comando DECODE, serve para comparar um valor e apresentar um outro. Vejamos a sua sintaxe e analisaremos o exemplo:

DECODE(valor, comparação1, valor1 [,comparaçãoN, valorN] [, valor_default)

No exemplo apresentado na Figura 10, a coluna de comparação é o department_id, estamos verificando se o valor dela é 10, se sim, ao invés de apresentar o conteúdo da tabela (Administration) iremos apresentar 'Gerencia', a próxima comparação é o valor 20, que será substituído pelo novo valor 'Propoganda', os demais valores serão os valores da tabela. Mantivemos na consulta os valores do department_id e department_NAME, para que possam ser analisados após a aplicação do DECODE.

FIGURA 10 – DECODE

```
select department_id,department_name,
DECODE(department_id, 10, 'Gerencia',20,'Propoganda',department_name)novo
from departments
```

Resultado da Consulta

Todas as Linhas Extraídas: 27 em 0,003 segundos

	DEPARTMENT_ID	DEPARTMENT_NAME	NOVO
1	10	Administration	Gerencia
2	20	Marketing	Propoganda
3	30	Purchasing	Purchasing
4	40	Human Resources	Human Resources

FONTE: O autor

Percebemos no resultado da execução, que somente os valores definidos na comparação são alterados.

4.2 CASE SIMPLES

A função Case faz parte do SQL/92 e por isso seu uso é mais indicado. Segundo Price (2009), existem dois tipos de expressões CASE, as simples e as pesquisadas. As empressões CASE simples, que usam expressões para determinar o valor retornadom e possuem a seguinte sitaxe:

CASE expressão_pesquisa
WHEN expressão1 THEN resutado1
WHEN expressão2 THEN resutado2
..
WHEN expressãoN THEN resutadoN
Else resultado_padrão
end

Segundo Price (2009), a **expressão_pesquisa**, é a expresão que será avalidas, a **expressão1, expressão1... expressãoN** são as expressões a serem avaliadas em relação à **expressão_pesquisa.** Já as colunas **resutado1, resutado2 e resutadoN,** são os resultados respectivamentes, e **resultado_padrão** será a exceção quando não encontrar nenhum valor corresponde. Vejamos um exemplo:

FIGURA 11 – CASE

```
select department_id,department_name,
CASE department_id
WHEN 10 THEN 'Gerencia'
WHEN 20 THEN 'Propoganda'
ELSE department_name
END nome
from departments
```

Resultado da Consulta

Todas as Linhas Extraídas: 27 em 0,003 segundos

	DEPARTMENT_ID	DEPARTMENT_NAME	NOME
1	10	Administration	Gerencia
2	20	Marketing	Propoganda
3	30	Purchasing	Purchasing
4	40	Human Resources	Human Resources

FONTE: O autor

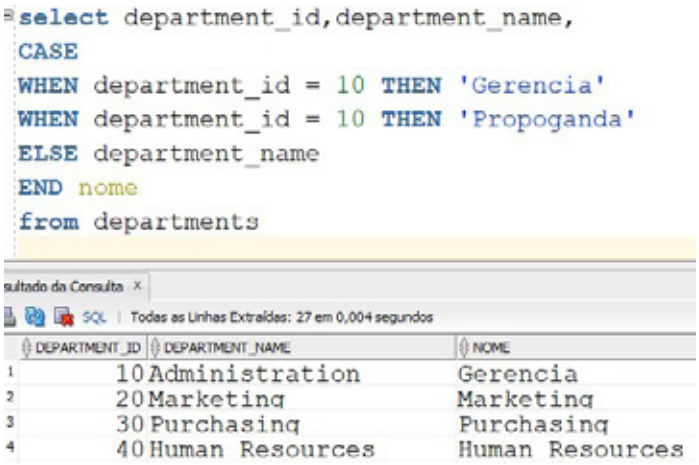
A função case, também pode ser realizada realizada de maneira pesquisada, comparando valores.

4.3 CASE PESQUISADA

Ainda segundo Price (2009), o a outro modelo de CASE é a CASE PESQUISADA, que utilizam condições para determinar o valor de retorno. Vejamos a sintaxe é muito semelhante, não tendo neste caso a expressão pesquisada:

```
CASE
WHEN expressão1 THEN resutado1
WHEN expressão2 THEN resutado2
..
WHEN expressãoN THEN resutadoN
Else resultado_padrão
End
```

FIGURA 12 – CASE PESQUISADA



FONTE: O autor

Como vantagem na CASE PESQUISADA, podemos além do comparativa "=", podemos usar outros comparativos e cálculos:

FIGURA 13 – EXEMPLO CASE PESQUISADA

```
select  salary,
CASE
WHEN salary/10 = 480 THEN 'pouco'
WHEN salary*2 < 48000  THEN 'muito'
ELSE ' ok'
END salarios
from employees
```

Resultado da Consulta

SQL | 50 linhas extraídas em 0,004 segundos

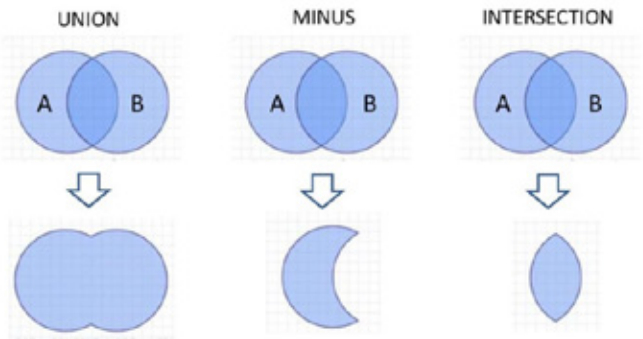
	SALARY	SALARIOS
1	24000	ok
2	17000	muito
3	17000	muito
4	9000	muito
5	6000	muito
6	4800	pouco

FONTE: O autor

5 OPERAÇÕES COM CONJUNTOS

Semelhante à teria dos conjuntos, muitas vezes precisamos buscar dados em tabelas diferentes, para isso podemos utilizar os comandos **UNION, INTERSECT E MINUS**.

FIGURA 14 – UNION, INTERSECT E MINUS



FONTE: Keryan (2015, s.p.)

Segundo Aléssio (2015), a união (UNION) de duas relações é o conjunto de todas as linhas que estão em uma ou outra relação, ignorando as duplicadas, ou seja, retorna à união de dois select's, ignorando as linhas que estão duplicadas. Já a interseção (INTERSECT) resultará no conjunto de todas as linhas que estão simultaneamente em ambas as relações, ou seja, retorna à interseção de dois select's. Já a diferença (MINUS) é o conjunto de todas as linhas que estão em apenas uma das relações, ou seja, retorna à subtração de dois select. Antes de apresentarmos os respectivos exemplos, vejamos algumas regras que devem ser obedecidas para a utilização desses comandos, segundo Aléssio (2015, p. 82).

- A cláusula Select deve selecionar o mesmo número de colunas.
- As colunas correspondentes devem ser do mesmo tipo de dado.
- As linhas duplicadas são automaticamente eliminadas.
- Os nomes das colunas do primeiro select é que aparecem no resultado.
- A cláusula order by deve aparecer no final do comando.
- A cláusula order by somente pode ser usada indicando o número da coluna.

Vamos verificar a figura, nela temos seleções muito parecidas, na quantidade e no tipo, mas não no conteúdo. Na primeira seleção, estamos buscando da visão que criamos anteriormente o salário anual, já na segunda estamos buscando o salário mensal. Também estamos indicando por qual coluna devemos fazer a ordenação, neste caso a coluna 3 que é a salário.

O comando Union não apresenta as linhas que se repetem, mas se você quiser apresentar as linhas que aparecem nas duas seleções, pode usar o UNON ALL, que segue as mesmas regras. Vejamos o exemplo:

```
SELECT 'A' pesquisa, v_emp.nome_completo nome, v_emp.salario_
anual salario, d.department_name
FROM V_Consulta_empregado v_emp, employees e, departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND v_emp.salario_anual > 200000
UNION
SELECT 'B' pesquisa, v_emp.nome_completo nome, e.salary salario,
d.department_name
FROM V_Consulta_empregado v_emp, employees e,departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND e.salary > 15000;
```

Veja que o primeiro valor das seleções são as letras A na primeira consulta, e a letra B na segunda, isso ajuda a identificar de qual seleção o dado está retornando.

Ao executamos o intersect, ele não retorna nenhuma linha, pois o resultado da coluna "pesquisa", evita que os dados sejam iguais, para a utilização desse comando, o ideal é que as colunas sejam as mesmas.

Exemplo Intersect:

```
SELECT 'A' pesquisa, v_emp.nome_completo nome, v_emp.salario_
anual salario, d.department_name
FROM V_Consulta_empregado v_emp, employees e, departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND v_emp.salario_anual > 200000
INTERSECT
SELECT 'B' pesquisa, v_emp.nome_completo nome, e.salary salario,
d.department_name
FROM V_Consulta_empregado v_emp, employees e,departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND e.salary > 15000;
```

Na consulta a seguir, temos a aplicação do MINUS, que retorna o conjunto de todas as linhas que estão em apenas uma das relações.

Exemplo MINUS:

```
SELECT 'A' pesquisa, v_emp.nome_completo nome, v_emp.salario_
anual salario, d.department_name
FROM V_Consulta_empregado v_emp, employees e, departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND v_emp.salario_anual > 200000
MINUS
SELECT 'B' pesquisa, v_emp.nome_completo nome, e.salary salario,
d.department_name
FROM V_Consulta_empregado v_emp, employees e,departments d
WHERE v_emp.cod_empregado = e.employee_id
AND d.department_id = e.department_id
AND e.salary > 15000;
```

Na próxima seção, conheceremos os subselects, que são utilizando quando o retorno de uma seleção é aplicado como filtro em outra seleção.

6 VISÕES – VIEW

Segundo Fanderuff (2003), uma visão ou *VIEW* é uma tabela lógica e como vantagem, ela não ocupa espaço no banco, uma vez que seus dados não estão fisicamente armazenados, ela é o resultado de uma seleção (consulta).

> De maneira geral, em uma visão é permitido selecionar, atualizar, exluir e incluir dados. No entanto, visões que contenham JOIN, GROUP BY, DISTINCT, aliases (para as colunas) e expressões somete permitem seleções (FANDERUFF, 2003, p. 193).

Camargo (2011) aponta algumas vantagens no uso de visões como reuso, segurança e simplificação de código. No reúso, elas são objetos de caráter permanente e podem ser lidas por vários usuários simultaneamente. No item segurança, permitem a ocultação de determinadas colunas de uma tabela. E na simplificação do código, permitem a criação de códigos de programação muito mais limpo.

A estrutura para a criação ou substituição de uma VIEW já existente. O comando REPLACE, irá substituir a visão caso ela já exista, por isso se deve ter cuidado ao usá-lo:

CREATE [OR REPLACE] VIEW nome_da_view [nome_da_coluna ,nome_da_coluna]] AS SELECT...

Exemplo de criação de uma VIEW:

```
CREATE OR REPLACE VIEW V_Consulta_empregado  AS
  SELECT  e.employee_id cod_empregado,
  e.first_name || e.last_name nome_completo,
  e.salary * 12 salario_anual,
  nvl(e.commission_pct,0) comissao,
  d.department_name departamento,
  j.job_title cargo,
  l.city cidade,
  c.country_name pais,
  trunc(MONTHS_BETWEEN(SYSDATE,hire_date)/12) tempo_empresa
FROM
  employees e,
  departments d,
  jobs j,
  locations l,
  countries c,
  regions r
```

```
WHERE e.department_id = d.department_id
  AND d.location_id = l.location_id
  AND l.country_id = c.country_id
  AND c.region_id = r.region_id
  AND j.job_id = e.job_id;
```

Após a criação da VIEW, ela pode ser consultada como qualquer outra tabela. Na criação desta visão, utilzando vários comandos, como: alias de colunas e tabelas, concatenação, e funções aninhadas que é utilizar uma função dentro da outra, para calcular o número de anos, utilizamos a MONTHS_BETWEEN, que retorna o número de meses entre duas datas, depois dividimos por 12 e por último, truncamos para trazer o número inteiro de anos.

Vale reforçar, que quando for consultar a visão, o usuário ou outro desenvolvedor não terá acesso às formulas, para ele será apenas uma tabela normal. É interessante, mas não obrigatório que você identifique a visão com um prefixo V_nome para saber que se trata de uma visão e que pode ter valores manipulados, como o nosso exemplo, a coluna salario_anual.

FIGURA 15 – CONSULTA VIEW

```
select * from V_Consulta_empregado
```

Resultado da Consulta

50 linhas extraídas em 0 segundos

	COD_EMPREGADO	NOME_COMPLETO	SALARIO_ANUAL	COMISSAO	DEPARTAMENTO	CARGO	CIDADE
1	100	StevenKing	288000	0	Executive	President	Seattle
2	101	NeenaKochhar	204000	0	Executive	Administratio...	Seattle
3	102	LexDe Haan	204000	0	Executive	Administratio...	Seattle
4	103	Alexander...	108000	0	IT	Programmer	Southlake
5	104	BruceErnst	72000	0	IT	Programmer	Southlake

CARGO	CIDADE	PAIS	TEMPO_EMPRESA
President	Seattle	United States of America	16
Administratio...	Seattle	United States of America	14
Administratio...	Seattle	United States of America	18
Programmer	Southlake	United States of America	14
Programmer	Southlake	United States of America	12

FONTE: O autor

As VIEWS, podem ser excluídas com o seguinte comando: **DROP VIEW nome_view;**

7 TABELA RESULTANTE

Em muitas situações quando estivermos testando uma procedure, queremos manter nossa base de dados integra e confiável, para isso, podemos criar uma tabela baseada em outra(s). Aproveitando sua estrutura e até mesmo seus dados. Segundo Fanderuff (2003), podemos criar a tabela levando apenas a estrutura, ou também os dados, o que determina é o resultado do select.

Sintaxe: **CREATE TABELA Nome_tabela (nome_coluna [restrições] [, nome_coluna [restrições]] [restrições]) as select ..**

Como podemos observar no exemplo 1, estamos criando uma tabela que manterá as mesmas colunas e os conteúdos da tabela origem. Mas em algumas situações, queremos apenas a estrutura, sem os dados, para isso podemos forçar que o select não retorne linha alguma, conforme o exemplo 2. Já no exemplo 3, temos um erro, pois estamos comparando um número com um texto, que não irá funcionar.

Uma dica muito importante é que um comando que não retorna linhas é diferente de um comando com problemas de sintaxe, ou seja, que não funciona.

```
/* Exemplo 1 Select com retorno*/
CREATE TABLE departamento_temp AS
SELECT department_id  cd_departamento
, department_name nm_departamento
, manager_id      cd_gerente
, location_id     cd_localizacao
FROM hr.departments;

/* Exemplo 2 Select com retorno*/
CREATE TABLE departamento_temp AS
     SELECT department_id   cd_departamento
        , department_name nm_departamento
        , manager_id      cd_gerente
        , location_id     cd_localizacao
     FROM hr.departments
  WHERE 5 = 6;
```

/* Exemplo 3 Select com erro, pois está comparando uma coluna numérica com um texto*/

```
CREATE TABLE departamento_temp AS
    SELECT department_id   cd_departamento
        , department_name nm_departamento
        , manager_id      cd_gerente
        , location_id     cd_localizacao
    FROM hr.departments
    WHERE location_id = 'teste';
```

Importante lembrar, que diferente da visão que não é criada fisicamente, a tabela resultante é criada no banco e para a sua alteração/manutenção é necessário dropar e repetir o novo processo. Para apagar a tabela, Sintaxe: drop table nome_tabela;

Mas muita atenção, pois o comando DROP não pode ser desfeito.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- Podemos manipular a forma de apresentação dos dados, através de máscaras.
- A apresentação dos dados como resultado, ficam mais claros quando utilizamos alias.
- Sempre que utilizamos tabelas com colunas com o mesmo nome, temos que utilizar alias de tabela.
- Funções como CASE e DECODE nos permitem testar e trabalhar os dados em tempo de execução da consulta.
- Podemos unir vários selects, mas que eles precisam ter a mesma quantidade de colunas e tipos de dados.
- Podemos apresentar o resultado de consultas, sem apresentar como esses dados foram trabalhados ou obtidos, com as views.

AUTOATIVIDADE

Com base no modelo de dados criado na Unidade 3, vamos realizar as nossas atividades. Vamos relembrar o modelo:

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

FONTE: Bezerra (2012, s.p.)

1 Apresente o nome do funcionário e o mês de admissão por extenso e em letra maiúscula exemplo: "LUIS SAMPAIO AGOSTO".

2 Apresente uma consulta, selecionando o nome do funcionário a descrição do seu cargo utilizando apelidos nas colunas e tabelas.

3 Desenvolva uma consulta que apresente a seguinte pesquisa:

a. O nome do funcionário com a primeira letra maiúscula.
b. O nome do departamento
c. O salário atual

d. Caso o salário seja menor que 500,00 apresentar um aumento de 10%.
e. Todas as colunas devem ter apelidos.

4 Desenvolva uma consulta que apresente o aumento de salário utilizando UNION. E deve apresentar os seguintes itens:

a. O nome do funcionário.
b. O nome do cargo.
c. O salário atual com o apelido "antigo".
d. Realizar a seguinte verificação, para os funcionários que ganham menos de R$400,00 apresentar um aumento de 30%. Os que ganham R$800,00 apresentar com 20% para os demais apenas 10%. Essa coluna deve ter o apelido "novo".

5 Com base na consulta realizada no exercício 4, criar uma visão chamada "novo_salario'.

TÓPICO 2

PL/SQL, BLOCOS ANÔNIMOS E EXCEÇÕES

1 INTRODUÇÃO

Nas unidades anteriores, vimos que podemos trabalhar as informações através das máscaras e dos alias para melhor visualização. Mas, nem sempre isso basta, algumas vezes é necessário desenvolver procedimentos.

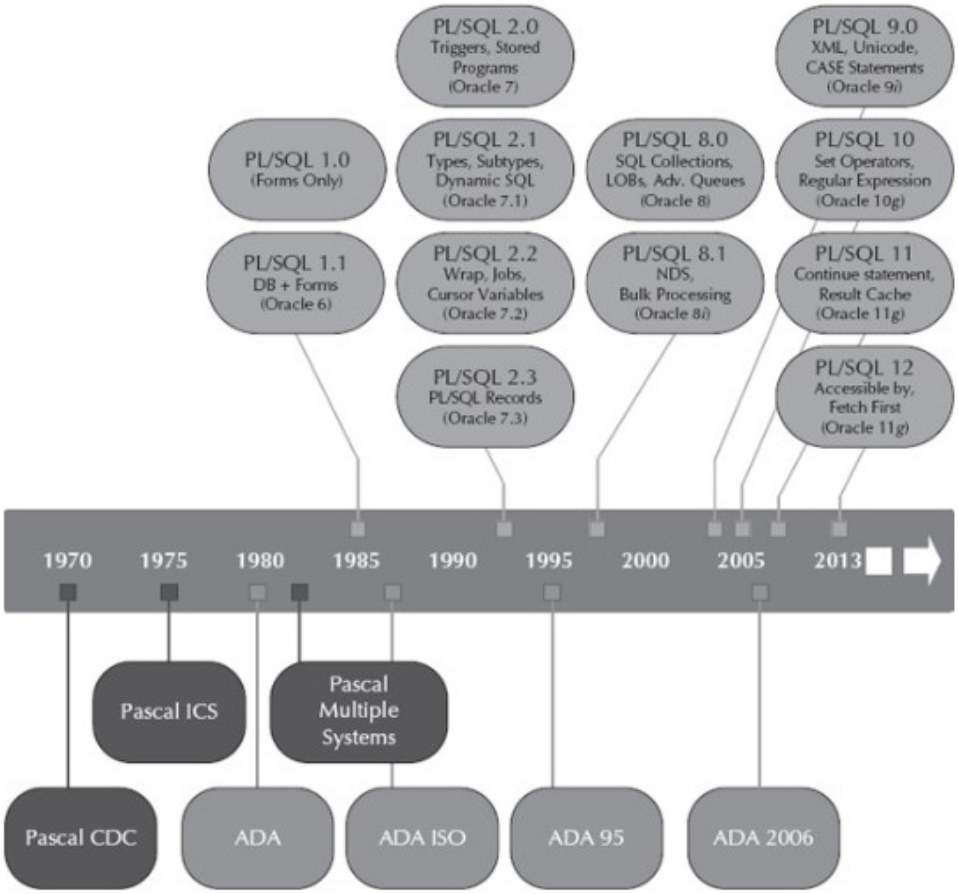
Estes procedimentos e funções, nos permitem realizar estruturas de repetições, como verificar todos os valores de uma tabela, também realizar testes, e com base nesses testes, tomar decisões como fazer uma inserção ou uma deleção, conforme regras de negócios.

Neste tópico, iremos falar da linguagem de programação PL/SQL e como ela é utilizada dentro do banco de dados Oracle. Começaremos conhecendo sua evolução.

2 PL/SQL

Segundo Mclaughlin (2014), a Oracle inovou além da especificação original do SQL e criou a sua própria linguagem PL/SQL, que é uma linguagem procedural que tem toda a flexibilidade do SQL. Embora muitos dos novos recursos do PL/SQL tenham sido adotados pelo padrão ANSI 92 SQL, alguns permanecem proprietários da Oracle. Esses recursos proprietários oferecem à Oracle uma vantagem competitiva. Da mesma forma, a Oracle atualmente define o padrão do setor para bancos de dados relacionais e objeto-relacionais. Vejamos a evolução através da figura a seguir.

FIGURA 16 – LINHA DO TEMPO DA LINGUAGEM PL/SQL



FONTE: Mclaughlin (2014, s.p.)

Atenção, pois esta parte do livro trata exclusivamente da Linguagem de Programação Procedural chamada PL/SQL. Existe um software chamado "PL/SQL Developer", mas que por sua vez não é a linguagem de programação que estamos estudando, trata-se de uma ferramenta. Acesse: https://www.oracle.com/br/tools/technologies/howto-sql--worksheet-insert-update-datele.html e https://www.youtube.com/watch?v=jZ19rKuRsSk.

Vamos agora conhecer e pôr em prática a programação em banco de dados, e utilizar os comandos que conhecemos nas outras unidades. Ela é bem simples e se assemelha à programação estruturada, ou à lógica de programação que aprendendemos no início da programação em algoritmos.

De acordo com Morais (2001), a unidade básica em um programa PL/SQL é um bloco, pois todos os programas da PL/SQL são compostos por blocos, que podem ser aninhados dentro do outro. Todos os programas da PL/SQL são compostos por blocos, que podem ser aninhados dentro do outro. Em geral, cada bloco realiza uma unidade lógica de trabalho no programa, assim separando um do outro diferentes tarefas.

Essa é a estrutura dos blocos PL/SQL, segundo Morais (2001):

```
DECLARE
/* variáveis, tipos, cursores e subprogramas locais */
BEGIN
/* Seção executável - instruções SQL e procedurais entram aqui. Essa é a principal sessão do bloco PL/SQL, e é a única obrigatória. */
EXCEPTION
/* Seção de tratamento de exceções – instruções de tratamento de erros entram aqui. */
END;
```

Podemos dizer que os blocos PL/SQL se dividem em três grupos, que detalharemos a seguir**: Blocos anônimos, Subprogramas e Triggers**. Sendo os subprogramas divididos em (PROCEDURES, FUNCTIONS e PACKAGES), que iremos estudar no Tópico 3 desta unidade. Antes, vamos conhecer um pouco algumas características dos Bloco Anônimo e Subprogramas.

Pesquise e conheça os triggers em: https://bit.ly/2RxXfHG.

QUADRO 3 – COMPARAÇÃO ENTRE BLOCO ANÔNIMO E SUBPROGRAMAS

	BLOCO ANÔNIMO	SUBPROGRAMAS
Compilação	A cada execução	Uma única vez
Possível colocar nome nos Blocos PL/SQL	Não	Sim
É reutilizável	Não	Sim
Pode ser armazenado no banco de dados?	Não	Sim
Pode ser chamado por outras programas, aplicações, subprogramas, scripts, etc.	Não	Sim, desde que esteja no banco de dados, ou no mesmo programa
Permite retornar valores	Não	Sim, inclusive as funções que são subprogramas devem retornar valores

FONTE: O autor

Vamos começar pelos blocos anônimos, eles serão a base para evoluirmos para os subprogramas.

3 BLOCOS ANÔNIMOS

Os blocos anônimos são blocos PL/SQL executáveis e sem nome, ou seja, são códigos que não são reutilizáveis e não podem ser armazenados no banco de dados para uso posterior. Vejamos a sua sintaxe.

Sintaxe:

```
DECLARE
  --declaração de variáveis caso necessário
BEGIN
  --bloco de desenvolvimento
END;
```

Talvez, você esteja se perguntando, qual é o motivo de criar um bloco anônimo, se ele não será armazenado para uso posterior. Ele é muitas vezes utilizado para o teste de procedures e funções. Ou quando queremos testar a nossa lógica, antes de criar um objeto no banco. Vejamos um exemplo simples, de um laço de repetição, que irá executar 9 vezes.

```
DECLARE
  v_nr_contador  NUMBER :=1;
BEGIN
WHILE (v_nr_contador < 9) LOOP
   dbms_output.put_line(v_nr_contador);
   v_nr_contador:=v_nr_contador+1;
END LOOP;
   dbms_output.put_line('Programa concluído');
END;
```

Como já mencionado, eles servirão de base para a criação de procedures e funções, ou para uma manutenção pontual no banco, e para a chamada de procedures. Como quase todo tipo de programação, precisamos de variáveis. Então, vamos começar com a definição da declaração das variáveis e constantes.

4 DECLARAÇÃO DE VARIÁVEIS E CONSTANTES

Segundo Aléssio (2015), todos os objetos declarados em uma seção DECLARE são locais ao bloco, e podem armazenar valores temporariamente e serem atualizados a qualquer momento. Lembrando que o <tipo> é o tipo de dados que será armazenado na variável, por exemplo, CHAR, NUMBER, DATE, BOOLEAN (True ou False).

QUADRO 4 – TIPO DE DADOS PARA VARIÁVEIS E CONSTANTES

Tipo		Conteúdo
BOOLEANO		Armazena valores TRUE e FALSE
DATE		armazena uma data válida
BLOB		Armazena uma imagem (uma fotografia por exemplo)
LONG RAW		Armazena um texto longo (um discurso por exemplo)
BFILE		Armazena um filme
VARCHAR2		Armazena um texto de tamanho variável
CHAR		Armazena um texto de tamanho fixo

FONTE: Adaptado de Moraes (2015)

Uma característica muito importante é a declaração das variáveis numéricas, vejamos como são definidos.

Sintaxe: **<nome_variável> <tipo> (<tamanho>[,decimais>]);**

QUADRO 5 – DECLARAÇÃO CAMPOS NUMÉRICOS

Exemplos	
V_compra number(5);	A variável v_compra, recebe o valor inteiro de 5 posições.
V_pagamento number(5,2);	A variável V_pagamento, recebe o valor de no máximo de 5 posições, sendo 3 inteiras e duas decimais. 999,99

FONTE: O autor

Para atribuir valores a uma variável, utiliza-se o comando de atribuição ":=", e finaliza-se com o ponto e vírgula.

Ex:= V_pagamento := 657, 34;

Reforçando que no PL/SQL, todo o comando SELECT pede a cláusula INTO para associar valores de colunas da base Oracle a variáveis PL/SQL.

Sintaxe: **SELECT** colunas **INTO** variáve**is FROM** tabelas **WHERE** condições;

Segundo Aléssio (2015), podemos utilizar as características de uma coluna de uma tabela utilizando o %TYPE, neste caso ele herda o tipo de dado da definição. No exemplo a seguir, a variável terá o mesmo tipo da tabela **EMPLOYEES** e da coluna **FIRST_NAME**. Vejamos a sintaxe e um exemplo:

Sintaxe: **Nome_variável tabela.coluna%type;**
Exemplo: **v_nome_empregado employees.first_name%type;**

Ao utilizarmos o %TYPE, estamos utilizando o tipo específico de uma coluna. Quando quisermos utilizar o tipo de uma tabela inteira, Aléssio (2015) apresenta o %ROWTYPE, pois ele cria um registro completo, utilizando-se as características de uma tabela da base. É como se a estrutura da tabela fosse armazenada na variável. Vejamos um exemplo de utilização para apresentar o salário anual do empregador código 100, utilizando o %ROWTYPE. Se fosse utilizar o %TYPE, precisaria definir as duas variáveis individualmente.

EXEMPLO:

```
DECLARE
  aux_empregado employees%ROWTYPE;
BEGIN
   SELECT * INTO aux_empregado
   FROM employees
   WHERE employee_id = 100;

   DBMS_OUTPUT.PUT_LINE('Salario annual:'
                ||to_char(aux_empregado.salary * 12)
                ||' para'
                ||aux_empregado.first_name);
END;
```

4.1 CONSTANTES

Semelhante ao conceito que já temos de constantes apresentado por Aléssio (2015), a autora afirma que elas são objetos que, uma vez declarados, não podem ter seus valores alterados. Portanto, a constante sempre recebe um valor no

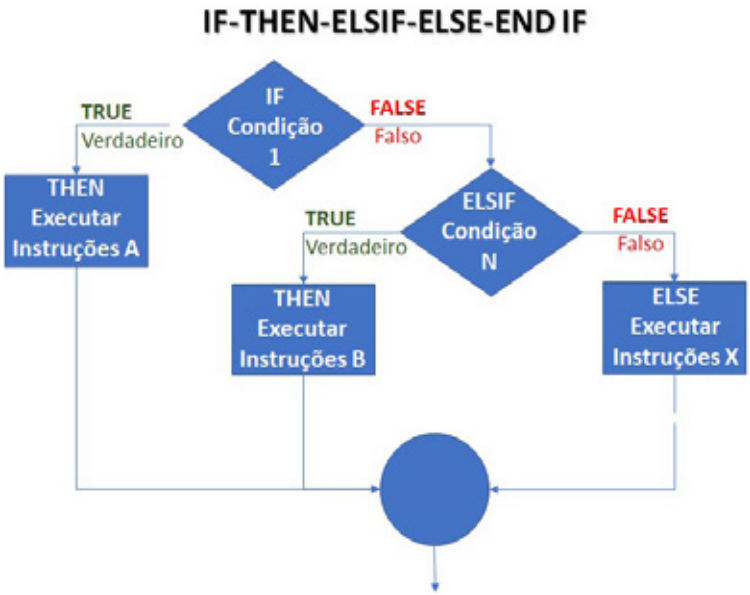
momento da declaração. Utiliza-se a palavra CONSTANT antes de se especificar o tipo. Vejamos o exemplo apresentado anteriormente, apenas definindo o número de meses como uma constante.

```
DECLARE
  aux_empregado employees%ROWTYPE;
  anual CONSTANT number(2) := 12;
BEGIN
  SELECT * INTO aux_empregado
  FROM employees
  WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('Salario de '
                ||to_char(aux_empregado.salary * anual)
                ||' para '
                ||aux_empregado.first_name);
END;
```

5 ESTRUTURAS DE CONTROLE

Semelhante à programação, dentro de uma estrutura PL/SQL, muitas vezes também temos que decidir, qual caminho seguir, para isso, podemos utilizar as estrutura IF..THEN. Segundo Dionísio (2015), esses comandos servem para definir durante a execução de código qual caminho seguir, ou seja, para quando uma condição for satisfeita. Esses testes são executados em uma dada sequência de instruções, onde temos três formas de apresentação desta instrução: IF-THEN, IF-THEN-ELSE e IF-THEN-ELSIF, vejamos o fluxo apresentando na imagem.

FIGURA 18 – ESTRUTURA DE CONTROLE



FONTE: O autor

As tabelas que utilizaremos nos exemplos, fazem parte do modelo de dados HR da Oracle, e do modelo de dado que criamos na Unidade 2".

Apresentamos algums exemplos, aplicando as três opções juntamente com a sintaxe dos comandos são:

QUADRO 6 – ESTRUTURAS DE CONTROLE - IF

	Instrução	Exemplo

IF-THEN.	IF condição THEN
	{instruções quando a condição for verdadeira}
	END IF;
		SET SERVEROUTPUT ON
		DECLARE
		v_salario_anual **NUMBER**(10,2) := 0;
		v_cod_func employees.employee_id%**TYPE**:=100;
		BEGIN
		SELECT salary*12 **INTO** v_salario_anual
		FROM employees
		WHERE employee_id = v_cod_func;
		IF v_salario_anual >= 150000 **THEN**
		DBMS_OUTPUT.PUT_LINE('Salário muito alto:'||v_salario_anual);
		END IF;
		END;

IF-THEN-ELSE	IF condição THEN
	{instruções quando a condição for verdadeira}
	ELSE
	{instruções quando a condição for falsa}
	END IF;
		SET SERVEROUTPUT ON
		DECLARE
		v_salario_anual **NUMBER**(10,2) := 0;
		v_cod_func employees.employee_id%**TYPE**:=100;
		BEGIN
		SELECT salary*12 **INTO** v_salario_anual
		FROM employees
		WHERE employee_id = v_cod_func;
		IF v_salario_anual >= 150000 **THEN**
		DBMS_OUTPUT.PUT_LINE('Salário muito alto:'||v_salario_anual);
		ELSE
		DBMS_OUTPUT.PUT_LINE('Salário muito baixo:'||v_salario_anual);
		END IF;
		END;

IF-THEN-ELSIF	IF condition1 THEN {instruções quando a condição for verdadeira} ELSIF condition2 THEN {instruções quando a condição for verdadeira} END IF;	**SET SERVEROUTPUT ON**
		DECLARE
		v_salario_anual **NUMBER**(10,2) := 0;
		v_cod_func employees.employee_id**%TYPE**:=100;
		BEGIN
		SELECT salary*12 **INTO** v_salario_anual
		FROM employees
		WHERE employee_id = v_cod_func;
		IF v_salario_anual = 150000 **THEN**
		DBMS_OUTPUT.PUT_LINE('Salário muito alto:'||v_salario_anual);
		ELSIF v_salario_anual = 25000 **THEN**
		DBMS_OUTPUT.PUT_LINE('Salário = 2500:'||v_salario_anual);
		ELSE
		DBMS_OUTPUT.PUT_LINE('outro salário :'||v_salario_anual);
		END IF;
		END;

CASE	CASE [expression]
	WHEN condition_1 THEN result_1
	WHEN condition_2 THEN result_2
	...
	WHEN condition_n THEN result_n
	ELSE result
	END
		SET SERVEROUTPUT ON
		DECLARE
		v_salario_anual **NUMBER**(10,2) := 0;
		v_cod_func employees.employee_id**%TYPE**:=100;
		BEGIN
		SELECT salary*12 **INTO** v_salario_anual
		FROM employees
		WHERE employee_id = v_cod_func;
		CASE v_salario_anual
		WHEN 150000 **THEN DBMS_OUTPUT.PUT_LINE**('Salario muito alto:'||v_salario_anual);
		WHEN 25000 **THEN DBMS_OUTPUT.PUT_LINE**('Salario = 2500:'||v_salario_anual);
		ELSE DBMS_OUTPUT.PUT_LINE('outro salario :'||v_salario_anual);
		END CASE;
		END;

FONTE: Adaptado de Dionisio (2015)

Agora que já vimos os testes, você percebeu que se assemelham à lógica da programação estruturada. Vamos conhecer os controles interativos, que você também já deve ter visto.

6 CONTROLE INTERATIVO

Os controles interativos, utilizados nos blocos PL/SQL, são os mesmos que já conhecemos da programação estruturada. Segundo Aléssio (2015):

QUADRO 7 – SINTAXE DOS COMANDOS DE CONTROLE INTERATIVO

Comando	Sintaxe
LOOP: O comando LOOP é utilizado para executar uma relação de comandos até que a condição definida na saída se torne verdadeira.	**LOOP** Relação_de_Comandos; **END LOOP;**
WHILE: O Comando WHILE associa uma condição à execução das iterações, sendo esta condição avaliada a cada iteração. Uma nova iteração só é iniciada se a condição for verdadeira.	**WHILE** <condição> **LOOP** comandos; **END LOOP;**
FOR: Permite que o loop seja repetido em uma conhecida quantidade de iterações.	**FOR** <cont> **IN** [REVERSE] <min>..<max> **LOOP** comandos; **END LOOP;**

FONTE: Adaptado de Aléssio (2015)

Em muitos códigos PL/SQL ou mesmo instruções SQL, você já deve ter visto essa barra "/" após o ponto é virgula. Ela funciona como um *ENTER*, executando o próximo comando.

Que tal praticar um pouco? Vejamos um exemplo de cada controle interativo. Todos os exemplos apresentam a solução para o mesmo problema. Tente resolver e veja quantas linhas foram inseridas.

QUADRO 8 – EXEMPLO DE CONTROLE INTERATIVO

Exemplo: LOOP	**DECLARE**
	aux_empregado employees**%ROWTYPE**;
	anual **CONSTANT NUMBER**(2) := 12;
	v_counter **NUMBER**(1):=0;
	BEGIN
	SELECT * **INTO** aux_empregado
	FROM employees
	WHERE employee_id = 100;
	LOOP
	INSERT INTO tabela_tempo(cod_func, Sal_anual) **VALUES** (aux_empregado.employee_id, aux_empregado.salary * anual) ;
	v_counter := v_counter + 1 ;
	EXIT WHEN v_counter = 1 ;
	END LOOP;
	END;
	/
	SELECT * **FROM** tabela_tempo;

Exemplo: WHILE	**DECLARE**
	aux_empregado employees**%ROWTYPE**;
	anual **CONSTANT NUMBER**(2) := 12;
	v_counter **NUMBER**(1):=1;
	BEGIN
	SELECT * **INTO** aux_empregado
	FROM employees
	WHERE employee_id = 100;
	WHILE v_counter <= 2 **LOOP**
	v_counter := v_counter+ 1 ;
	INSERT INTO tabela_tempo(cod_func, Sal_anual) **VALUES** (aux_empregado.employee_id, aux_empregado.salary * anual) ;
	v_counter := v_counter + 1 ;
	END LOOP;
	END;
	/
	SELECT * **FROM** tabela_tempo;

Exemplo: FOR	**DECLARE**
	aux_empregado employees%rowtype;
	anual CONSTANT number(2) := 12;
	BEGIN
	SELECT * **INTO** aux_empregado
	FROM employees
	WHERE employee_id = 100;
	FOR I **IN** 1..2 **LOOP**
	INSERT INTO tabela_tempo(cod_func, Sal_anual) **VALUES** (aux_empregado.employee_id+1, aux_empregado.salary * anual) ;
	END LOOP;
	END;
	/
	SELECT * **FROM** tabela_tempo;

Fonte: O autor

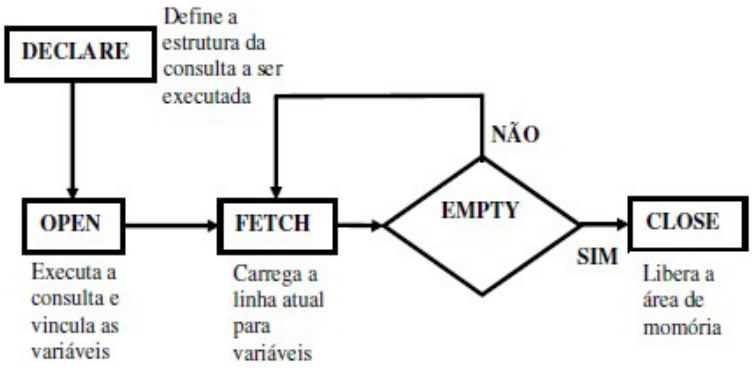
Para testar os exemplos criamos uma tabela temporária: Create table tabela_tempo (cod_func, Sal_anual) as select employee_id, salary*12 from employees where 5=6

No próximo item, conhecemos os cursores, que permitem que esses laços sejam baseados e select´s.

7 CURSORES

A utilização dos cursores facilita muito a programação PL/SQL, pois há momentos quando necessitamos de espaços de armazenamento mais complexos que as variáveis. Seria como uma matriz de informação, por exemplo, que é o resultado de uma consulta SQL. A isso chamamos de cursores e são divididos em implícitos ou explícitos. Segundo Willian (2009), o PL/SQL declara um cursor implicitamente para toda instrução DML (UPDATE, INSERT, DELETE), incluindo consultas (SELECT) que retornam apenas uma linha. Já nos casos onde as consultas que retornam mais de uma linha deverão ser declaradas explicitamente. Vale lembrar que os cursores explícitos são indicados quando é necessário um controle no processamento deles. Os cursores, são estruturas criadas para selecionar várias linhas de resultado, ou seja, são comandos SELECT que podem retornar mais de uma linha de resultado (ou nenhuma).

FIGURA 18 – CURSORES



FONTE: Moraes (2015, p. 109)

Segundo Moraes (2015), a definição de um cursor é:

- Sintaxe: DECLARE CURSOR nome_cursor IS comando_select Abertura: neste instante o comando SELECT declarado é executado e os dados ficam disponíveis na memória.
- Sintaxe: OPEN nome_cursor Busca de dados para recuperar as informações da linha onde está posicionado o ponteiro do cursor, utilizamos o comando FETCH. As variáveis utilizadas devem estar declaradas.
- Sintaxe: FETCH nome_cursor INTO lista_de_variáveis Fechamento
- Para fechar um cursor aberto e liberar a área de memória por ele utilizada usamos o comando CLOSE.
- Sintaxe: CLOSE nome_cursor.

Nos próximos itens conheceremos um pouco mais a fundo os cursores explícitos e implícitos.

7.1 CURSORES IMPLÍCITOS

Um cursor implícito é um cursor de sessão que é construído e gerenciado por PL/SQL, segundo (JAYAPALAN *et al*., 2019), ou seja, não temos controle sobre a forma que o banco de dados gerará construir o comando na memória. De forma simples, podemos dizer que o PL/SQL abre um cursor implícito toda vez que você executa uma instrução SELECT ou comandos como de INSERT, UPDATE, DELETE, MERGE. Você não pode controlar, mas pode obter informações de seus atributos.

Num cursor explícito escolhemos o nome e o utilizamos para consultar os atributos como visto no item de "Cursores Explícitos" deste livro. Já ao cursor implícito não podemos definir um nome, todos os cursores implícitos são chamados de SQL, por isso um cursor implícito também é chamado de Cursor SQL.

Na maioria dos exemplos apresentamos uma mensagem. Mas dependendo do cenário, você pode realizar um teste (IF/ELSE) e também poderia inserir/alterar/deletar em uma tabela.

QUADRO 7 - ATRIBUTOS DE CURSORES IMPLÍCITOS

Cursor Implícito	Retorno do Atributo
SQL%ISOPEN	Sempre retornará FALSE, porque um cursor implícito sempre fecha após a execução dele.
SQL%FOUND	• NULL: Se nenhuma instrução SELECT ou DML foi executada. • TRUE: Se a instrução mais recente SELECT ou DML retornou uma linha. • FALSE: Se a instrução mais recente SELECT ou DML não retornou uma linha.
SQL%NOTFOUND	• NULL: Se nenhuma instrução SELECT ou DML foi executada. • FALSE: Se a instrução mais recente SELECT ou DML retornou uma linha. • TRUE: Se a instrução mais recente SELECT ou DML não retornou uma linha.
SQL%ROWCOUNT	• NULL: Se nenhuma instrução SELECT ou DML foi executada. • Se uma instrução SELECT ou DML foi executada, o número de linhas buscadas até o momento.

FONTE: Jayapalan (2019)

Existem os atributos %BULK_EXCEPTIONS e %BULK_ROWCOUNT, consulte o artigo de Tércio Costa, 2016 "Entendendo e Utilizando os atributos SQL%BULK_ROWCOUNT e SQL%BULK_EXCEPTIONS" Disponível no site da Oracle em: https://bit.ly/2yZhoAf.

Aqui temos um exemplo de cursor implícito (**SQL%ROWCOUNT**), utilizado com o comando update. Neste exemplo, vemos que é possível saber quantos registros foram atualizados com a instrução SQL%ROWCOUNT. Para isso, criamos um bloco anônimo

```
SET SERVEROUTPUT ON
DECLARE
BEGIN
  UPDATE hr.employees
     SET salary = salary * 1.25
   WHERE salary < 5000;
DBMS_OUTPUT.PUT_LINE('Atualizado(s)'|| SQL%ROWCOUNT ||'
 Salário(s)');
END;
```

Neste caso, o resultado dessa execução irá apresentar a seguinte mensagem na tela do seu SGBD. Atualizado(s) 42 Salário(s), pois ele encontrou na base de dados, 42 registros que atendem à restrição de salário menor que 5000.

Dependendo da ferramenta que você estiver utilizando, é necessário habilitar o comando SET SERVEROUTPUT ON, pois ele imprime a saída criada pelo pacote DBMS_OUTPUT a partir dos métodos PL / SQL. O programa PL / SQL é executado no mecanismo Oracle, portanto, é necessário obter o resultado da saída do servidor e exibir na tela; caso contrário, os resultados não serão exibidos.

No exemplo a seguir, vamos utilizar o SQL%FOUND para verificar se o comando DELETE encontrou algum registro para excluir. Neste exemplo, para reforçar os conteúdos, estamos criando uma variável, usando o %TYPE. Lembrando que a variável VCD_LOCAL, terá mesmo tipo e tamanho da coluna cddepto da tabela departamentos. Vejamos o *Script de exemplo SQL%ROWCOUNT*:

```
DECLARE
  VCD_LOCAL departamentos.cddepto%TYPE;
BEGIN
 vcd_local := 'D6';
 DELETE FROM funcionario
 WHERE cddepto = vcd_local;
 IF SQL%FOUND THEN
   DBMS_OUTPUT.PUT_LINE ('Excluído com sucesso ' ||SQL%ROW
COUNT ||' departamento'|| CASE WHEN SQL%ROWCOUNT > 1
THEN 's' ELSE '' END ||' do local: ' || vcd_local);
 ELSE
   DBMS_OUTPUT.PUT_LINE ('Não existe o departamento para o local:
' || vcd_local);
 END IF;
END;
```

Neste exemplo, estamos apresentando uma função extra, a aplicação do CASE na mensagem com o SQL%ROWCOUNT, para personalizar a mensagem. Se o resultado do SQL%ROWCOUNT for maior que 1, vamos concatenar via a letra "s" a palavra departamento. Isso não é obrigatório, mas apresenta uma mensagem mais clara, ela poderia ser substituída por: **DBMS_OUTPUT.PUT_LINE** ('Excluído com sucesso ' ||**SQL%ROWCOUNT** ||' departamento');

7.2 CURSORES EXPLÍCITOS

Vimos como podemos tirar proveito dos cursos implícitos, entre outras coisas, para controlar as alterações no banco, mas existem situações em que precisamos definir nós mesmos os cursores, que são os explícitos, de acordo com Nuitjen *et al*. (2016, s.p.).

> Cursores explícitos são cursores que um programador definiu para ter mais controle sobre um cursor e a área de contexto. Um cursor explícito é definido na seção de declaração de um bloco PL/SQL e permite que você nomeie o cursor, e é por isso que os cursores explícitos também são chamados de cursores nomeados. Quando você nomear o cursor, poderá acessar sua área de trabalho e informações, bem como processar as linhas da consulta individualmente.

Um cursor explícito, segundo Nuitjen *et al.* (2016), é mais complicado que um cursor implícito, porque você precisa construí-lo e gerenciá-lo; no entanto, possui alguns benefícios que tornam todo o problema valioso. Ao declarar um cursor explícito, você o define, nomeia e associa-o a uma consulta. Em seguida, você pode usar operações como OPEN, FETCH e CLOSE com o cursor, ou usá-lo com uma instrução FOR LOOP.

Segundo Moraes (2015), alguns atributos do cursor podem ser testados durante a execução do programa, que são: %ISOPEN, %FOUND, %NOTFOUND, %NOTFOUND. Vejamos a seguir um quadro, que apresenta o nome do cursor e qual será o retorno dele, durante a execução.

QUADRO 8 – ATRIBUTOS DE CURSORES EXPLÍCITOS

Cursor Explícito	Retorno do Atributo
Nome_cursor%ISOPEN	**TRUE**: se o cursor explícito estiver aberto. **FALSE**: se o cursor explícito estiver fechado. **OBS:** este é o único atributo que pode ser testado antes de executar o comando OPEN nome_cursor;
Nome_cursor %FOUND	**NULL**: se nenhum FETCH for executado do cursor. **TRUE**: se o SELECT do cursor retornou alguma linha. **FALSE**: se o SELECT do cursor não retornou linhas.
Nome_cursor %NOTFOUND	**NULL**: se nenhum FETCH for executado no cursor. **FALSE**: se o SELECT do cursor retornou alguma linha. **TRUE**: se o SELECT do cursor não retornou linhas.
Nome_cursor %ROWCOUNT	**NULL**: se nenhum FETCH for executado do cursor. ***N*** – retorna o número de linhas trazidas pelo SELECT do cursor explícito.

FONTE: Adaptado de Moraes (2015)

Já vimos que existem várias formas de utilizar um cursor explícito. A seguir colocaremos algumas das sintaxes de cursor e em seguida no mesmo quadro um exemplo de como funciona esta sintaxe dentro do PL/SQL.

SINTAXE:

```
DECLARE
   /*Declarando o cursor */
   CURSOR c_nome_cursor IS
   SELECT {colunas}
    FROM {tabela}
   WHERE {condição};
```

```
   /*Declarando uma variável que será o registro da tabela*/
   r_nome_cursor c_nome_cursor%ROWTYPE;
BEGIN
   /* Abre cursor */
   OPEN c_nome_cursor
   LOOP
      /*Lê um registro do cursor*/
      FETCH c_nome_cursor INTO r_nome_cursor;
      /*Abandona o loop caso seja o final do cursor*/
      EXIT WHEN c_nome_cursor%NOTFOUND;
      /*Aqui será inserido o código que irá manipular os dados*/
   END LOOP;
   /*Fecha o cursor*/
   CLOSE c_nome_cursor
END;
```

A seguir temos um exemplo, onde fazemos um SELECT na tabela employees e departments, buscando o nome e o sobrenome (e.first_name, e.last_name) dos empregados que estão alocados no departamento 80 (e.department_id = 80). Perceba que estamos relacionando as duas tabelas através da chave primária com a chave estrangeira (e.department_id = d.department_id). Outro item importante para não dar erro é o *alias* de tabela. Como resultado, apresentamos uma mensagem (DBMS_OUTPUT.PUT_LINE), concatenando o um texto fixo (‘Nome: ’) com o resultado do cursor (r_employees);

Exemplo:

```
DECLARE
  CURSOR c_employees IS
       SELECT e.first_name, e.last_name
       FROM employees e,
       departments d
       WHERE e.department_id = d.department_id
       AND e.department_id = 80;
  r_employees c_employees%ROWTYPE;

BEGIN
  OPEN c_employees;
  LOOP
    FETCH c_employees INTO r_employees;
     EXIT WHEN c_employees%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Nome:||r_employees.first_name||' '||r_
employees.last_name);
  END LOOP;
 CLOSE c_employees;
END;
```

Continuando com o exemplo apresentado acima, onde utilizamos OPEN, vamos utilizar o mesmo exemplo aplicar em outro formato de cursor. Neste, a abertura do cursor é definida no DECLARE e não há a necessidade de criar a variável de registro para armazenar e nem abrir e fechar o cursor, tudo é feito automaticamente. Esse cursor é chamado de **FOR..LOOP**. A escolha de um ou outro não interfere no resultado. Vamos analisar a sintaxe:

```
DECLARE
   /*Declarando o cursor */
   CURSOR c_nome_cursor IS
        SELECT {colunas}
          FROM {tabela}
         WHERE {condição};
BEGIN
   FOR r_nome_cursor IN c_nome_cursor     LOOP
      /*Aqui será inserido o código que irá manipular os dados*/
   END LOOP;
END;
```

O código é mais simples, vejamos o mesmo exemplo apresentado acima, utilizando esta sintaxe. Neste exemplo iremos inserir em uma tabela chamada aumento_natal e apresentar uma mensagem do novo salário. Se o funcionário receber mais que 800, ele ganhará um aumento de 10%, se ganhar menos receberá 20%.

```
DECLARE
  CURSOR c_employees IS
SELECT e.first_name, e.last_name, e.salary
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND e.department_id = 80;
BEGIN
  FOR r_employees IN c_employees   LOOP
   IF r_employees.salary < 8000 THEN
   DBMS_OUTPUT.PUT_LINE(' Salario menor que 8000:  Nome: '
||r_employees.first_name ||' '|| r_employees.last_name||' '||r_
employees.salary*1.2);
    INSERT INTO aumento_natal VALUES(r_employees.first_name, r_
employees.last_name, r_employees.salary*1.1);
    ELSE
    DBMS_OUTPUT.PUT_LINE('Salario maior que 8000:Nome: '
||r_employees.first_name ||' '|| r_employees.last_name||' '||r_
employees.salary*1.1);
    INSERT INTO aumento_natal VALUES(r_employees.first_name, r_
employees.last_name, r_employees.salary*1.1);
  END IF;
```

```
 END LOOP;
END;
```

Vamos comparar os dois códigos e a escolha de um ou outro modelo, não irá interferir no resultado.

QUADRO 9 – COMPRANDO OS MODELOS

OPEN ..FETCH	FOR..LOOP

DECLARE
CURSOR c_employees IS
SELECT e.first_name, e.last_name
FROM employees e,
departments d
WHERE e.department_id = d.department_id
AND e.department_id = 80;
r_employees c_employees%ROWTYPE;
BEGIN
OPEN c_employees;
LOOP
FETCH c_employees INTO r_employees;
EXIT WHEN c_employees%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Nome:||r_employees.first_name);
END LOOP;
CLOSE c_employees;
END;
	DECLARE
	CURSOR c_employees IS
	SELECT e.first_name, e.last_name
	FROM employees e, departments d
	WHERE e.department_id = d.department_id
	AND e.department_id = 80;
	BEGIN
	FOR r_employees IN c_employees LOOP DBMS_OUTPUT.PUT_LINE('Nome: '||r_employees.first_name);
	END LOOP;
	END;

FONTE: O autor

Uma Subquery (também conhecida como SUBCONSULTA ou SUBSELECT) é uma instrução do tipo SELECT dentro de outra instrução SQL. Desta forma, se torna possível efetuar consultas que de outra forma seriam extremamente complicadas ou impossíveis de serem feitas de outra forma. Conheça um pouco mais sobre esse assunto e leia o artigo: https://www.devmedia.com.br/trabalhando-com-subqueries/40134.

Mas quando algum cursor não retornar linhas, ou acontecer algo inesperado no meu código. Para isso podemos tratar com exceções.

8 TRATAMENTO DE EXCEÇÕES

Em muitas situações, o código pode apresentar erros, seja por não encontrar um registro, ou por encontrar vários, conforme Jayapalan *et al.* (2019). Em um programa PL/SQL erros em tempo de execução são considerados exceções (***EX-***

CEPTION), pois podem surgir de falhas de design, erros de codificação, falhas de hardware e muitos outros tipos de erros. Você não pode prever todas as exceções possíveis, mas pode escrever manipuladores de exceções que permitam que seu programa continue a operar na presença deles. Qualquer bloco PL/SQL pode ter uma parte de tratamento de exceções. Seguindo a sintaxe:

```
DECLARE
 ....
BEGIN
 ...
EXCEPTION
  WHEN nome_excecao1 THEN
     instruções1
  WHEN nome_excecao2 OR nome_excecao3 THEN
     instruções2
  WHEN OTHERS THEN
     instruções3
END;
```

Existem várias exceções, vamos apresentar algumas no quadro a seguir. Listamos as algumas das exceções mais usadas, juntamente com seu nome abreviado e ORA - código de erro:

QUADRO 10 – EXCEÇÕES DEFINIDAS PELO SISTEMA

Erro (SQLERRM)	**Nome da Exceção**	**Quando**
ORA-00001	DUP_VAL_ON_INDEX	Existe um valor duplicado
ORA-01001	INVALID_CURSOR	O cursor é inválido
ORA-01012	NOT_LOGGED_ON	Usuário não está logado
ORA-01017	LOGIN_DENIED	Ocorreu um erro no sistema
ORA-01403	NO_DATA_FOUND	A consulta não retorna dados
ORA-01422	TOO_MANY_ROWS	Uma consulta de linha única retorna várias linhas
ORA-01476	ZERO_DIVIDE	Foi feita uma tentativa de dividir um número por zero
ORA-01722	INVALID_NUMBER	O número é inválido
ORA-06504	ROWTYPE_MISMATCH	Ocorreu uma incompatibilidade no tipo de linha
ORA-06511	CURSOR_ALREADY_OPEN	O cursor já está aberto
ORA-06531	COLLECTION_IS_NULL	Trabalhando com NULL em coleção
ORA-06532	SUBSCRIPT_OUTSIDE_LIMIT	Índice de coleção fora do intervalo
ORA-06533	SUBSCRIPT_BEYOND_COUNT	Índice de coleção fora da contagem

FONTE: Jayapalan *et al.* (2019)

As mensagens de erros do banco de dados Oracle 19c podem ser consultadas em: https://docs.oracle.com/en/database/oracle/oracle-database/19/errmg/.

Uma dica, é sempre usar como última exceção a ser testada a WHEN OTHERS, pois se ela foi colocada no começo, nenhuma das outras posteriores serão testadas em caso de erro. Seria como o ELSE dos testes condicionais.

As exceções, podem ser definidas internamente, segundo Gonçalves (2012, s.p.), em dois tipos de exceção:

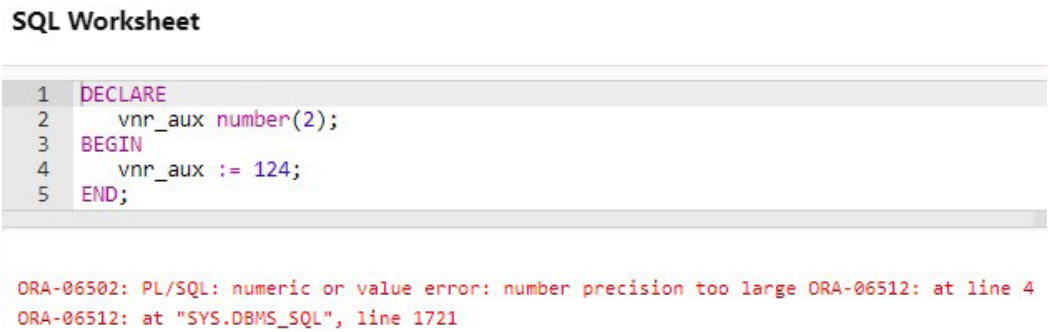
> **Exceção de sistema (implícita):** um erro definido pelo Oracle que é disparado automaticamente pela *runtime engine* da PL/SQL assim que ela detecta o problema. Exemplos: TOO_MANY_ROWS (retorno mais de uma linha) e NO_DATA_FOUND (nenhum dado encontrado).
> **Exceção Programada (explícita):** exceção definida pelo programador e, portanto, específica da aplicação. Este tipo de exceção não é disparado automaticamente, mas apenas para situações indicadas pelo programador em seu código.

Vamos detalhar um pouco melhor cada um dos tipos, começando com as exceções implícitas.

8.1 EXEMPLO DE TRATAMENTOS DE EXCEÇÕES DE SISTEMA (IMPLÍCITA)

Para facilitar o entendimento, vamos começar observando o código a seguir na Figura 20, que apresenta erro, pois iremos atribuir um valor maior que o tamanho da variável, e consequentemente irá gerar erro, neste exemplo não estamos fazendo nenhum tipo de tratamento de exceção.

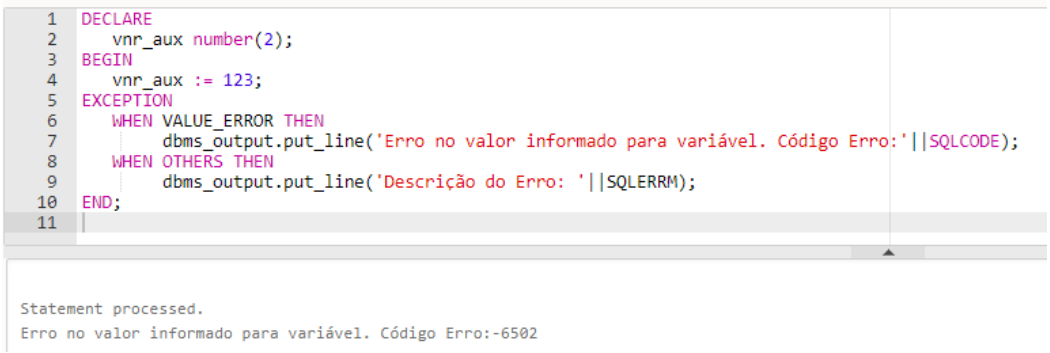
FIGURA 19 – EXECUÇÃO BLOCO PL/SQL SEM TRATAMENTO DE EXCEÇÃO



FONTE: O autor

Este erro não é uma situação muito extraordinária, podemos dizer que ele é até comum de acontecer. Vamos agora colocar "tratamento de exceção" no exemplo apresentado acima. Ao tratar a exceção, ao invés do sistema dar erro, ele é direcionado para a instrução programada, neste caso uma mensagem, mas poderia ser qualquer outra coisa, como o inserte em uma tabela de log. Uma dica muito importante é que você deve sempre que possível tratar os erros.

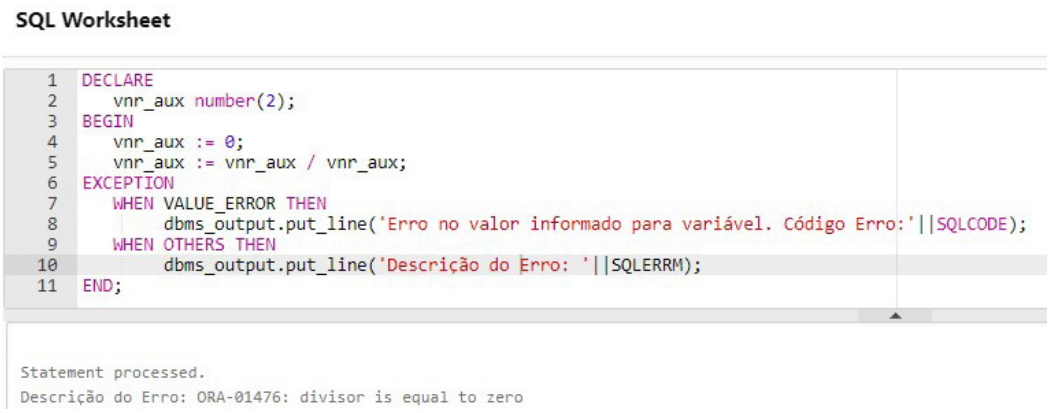
FIGURA 20 – ATRIBUINDO NÚMERO MAIOR QUE O TAMANHO DECLARADO (COM TRATAMENTO DE EXCEÇÃO)



FONTE: O autor

Vamos acompanhar mais um exemplo de aplicação de EXCEPTION, no exemplo da figura a seguir, forçamos um erro através de uma divisão por zero, que não é permitido pela matemática. Veja que a exceção "VALUE_ERROR" já não funciona para divisão por zero. Poderíamos ter colocado a exceção "ZERO_DIVIDE", que é específica para o caso, mas deixamos OTHERS para exemplificar que qualquer exceção não definida, cairá em OTHERS, por isso é importante que esta seja a última opção das exceções.

FIGURA 21 – FORÇANDO DIVISÃO POR ZERO (COM TRATAMENTO DE EXCEÇÃO)



FONTE: O autor

Perceba que nos exemplos que acabamos de estudar, aparecem "SQLCODE" e "SQLERRM". Segundo Gupta (2016), o Oracle fornece duas funções utilitárias: SQLCODE retorna o código de erro e SQLERRM retorna a mensagem da exceção mais recente.

No exemplo a seguir, utilizamos o mesmo código para analisar dois cenários diferentes. Na instrução (location_id = {PARAMETRO}), vamos substituir o {PARAMETRO} por 1700, e vamos testar. Neste cenário, não teremos retorno. Já se utilizarmos o o {PARAMETRO} com 1800, ele retornará o Departamento: 20 Marketing, porque não encontrou nenhuma linha.

QUADRO 12 – EXEMPLO EXCEÇÃO NO_DATA_FOUND E TOO_MANY_ROWS

```
DECLARE
  vnm_departamento hr.departments.department_name%TYPE;
  vcd_departamento hr.departments.department_id%TYPE;
BEGIN
  SELECT department_name, department_id
   INTO vnm_departamento, vcd_departamento
   FROM hr.departments
   WHERE location_id = {PARAMETRO};
   dbms_output.put_line('Departamento: '||vcd_departamento
                  ||' '||vnm_departamento);
EXCEPTION WHEN NO_DATA_FOUND THEN
    dbms_output.put_line('Departamento:'||vcd_departamento
                    ||' não foi encontrado');
   WHEN TOO_MANY_ROWS THEN
    dbms_output.put_line('Muitos registros consulta');
END;
```

Se colocar em {PARAMETRO}	Resultado da execução
1700	Retornaram muitos registros para esta consulta
1800	Departamento: 20 Marketing

FONTE: O autor

Existem situações em que eu quero controlar o erro e não parar o programa, utilizamos as exceções explícitas. Que são definidas pelo desenvolvedor, que são RAISE e RAISE_APPLICATION_ERROR.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- Existem muitas vantagens na utilização da linguagem PL/SQL.
- Como fazer testes condicionais e como utilizar as estruturas de controle.
- Há dois cursores: implícitos e explícitos.
- Como as exceções são importantes para garantir a qualidade do nosso código.

AUTOATIVIDADE

Dando sequência às autoatividades, utilizaremos o mesmo modelo de dados apresentado anteriormente.

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

1 Criar um bloco anônimo para apresentar a soma do salário anual dos funcionários do departamento 'C3'. Se a soma do salário anual for menor que 30.000 deverá apresentar uma mensagem dizendo "Salário muito baixo:" concatenando o valor do salário. Caso contrário, apresentar a seguinte mensagem 'Salário muito alto:" e o valor do salário.

2 Usando como cenário do exercício anterior, vamos criar um bloco anônimo para aumentar em 10% os salários dos funcionários do departamento de vendas (D5) e criar uma mensagem personalizada.

Dica: Após a alteração, você deve verificar a quantidade de registros alterados e padronizar a mensagem, para isso utilizar o SQL%FOUND e o SQL%ROWCOUNT.

3 Seguindo o modelo apresentado no exercício 1, onde selecionamos apenas o departamento C3, agora iremos utilizar a estrutura de cursores OPEN FETCH. Para isso, desenvolva um bloco anônimo que apresente o salário anual de todos os departamentos. E ao final, uma mensagem informando o nome e o salário anual do departamento.

4 Desenvolva o mesmo exercício número 3, agora utilizando o FOR LOOP. Mas ao invés de apresentar uma mensagem, insira os dados em uma tabela. A tabela criada terá o nome de Resultado_anual.

5 Refaça o exercício número 4, só que agora utilizando duas EXCEPTION, que se adequem ao insert.

TÓPICO 3

PROCEDURES E FUNCTIONS

1 INTRODUÇÃO

Antes de começarmos os nossos estudos, vamos rever a principal diferença entre procedures e funções. Uma procedure, pode conter uma lista de argumentos, e pode retornar um ou mais valores, já a função pode conter uma lista de argumentos e deve retornar apenas um valor. Ou seja, o retorno na procedure é opcional e na função é obrigatório.

Vamos começar comparando as duas estruturas, que são muito semelhantes:

QUADRO 12 – SINTAXE DE CRIAÇÃO PROCEDIMENTOS E FUNÇÕES

Objeto	Sintaxe
Procedure	CREATE OR REPLACE PROCEDURE nome da procedure (argumento1 modo tipo de dado , argumento2 modo tipo de dado ,argumentoN modo tipo de dado) IS ou AS declaração das variáveis locais, constantes, etc. BEGIN /*corpo da procedure que define a ação executada*/ END nome da procedure;
Function	CREATE OR REPLACE FUNCTION nome da função (argumento1 modo tipo de dado ,argumento2 modo tipo de dado ,argumentoN modo tipo de dado) RETURN tipo de dado (retorna apenas um parâmetro) IS ou AS declaração variáveis locais, constantes, etc. BEGIN /*corpo da função que define a ação executada*/ RETURN [valor que será retornado]; END nome da função;

FONTE: Adaptado de Aléssio (2015)

Agora que já conhecemos as similaridades e diferenças entre procedures e funções, vamos começar a desenvolver.

2 *PROCEDURES*

De acordo com Dionisio (2015), uma *stored procedure* é um bloco de instruções PL/SQL, que executa uma ou mais tarefas específicas. Elas são bem similares com as procedures de outras linguagens de programação. Para que possamos criar uma procedure no PL/SQL precisamos entender o seu funcionamento e para isso precisamos entender a sua sintaxe básica.

QUADRO 13 – SINTAXE DE PROCEDURE

CREATE [OR REPLACE] PROCEDURE nome_da_procedure [(nome _parâmetro [IN | OUT | IN OUT] tipo_dado_parâmetro [, ...])] {**IS**} Seção de declarações **BEGIN** Seção de execução **EXCEPTION** Seção de exceções **END** [nome_da_procedure];	
Partes da sintaxe	**Qual a função**
CREATE [OR REPLACE] PROCEDURE	Instrução que irá criar (***CREATE***) ou atualizar (**REPLACE**) um procedimento caso esta já exista com o mesmo nome.
nome_da_procedure	Nome que será atribuído para o procedimento
nome_parâmetro	Nome do parâmetro que está sendo passado para a procedure. Podem ser passados N parâmetros para uma procedure, apenas separá-los por vírgula. Um parâmetro pode ser de três tipos: • Parâmetro de entrada (IN): quando é passado um valor para ser utilizado na procedure. • Parâmetro de saída (OUT): quando é deixado um parâmetro para retornar um valor da procedure. • Parâmetro de entrada e saída (IN OUT): quando é passado um valor para a procedure e esse pode ser alterado devolvido para o parâmetro;
tipo_dado_parâmetro	É o tipo de retorno que será utilizado, sendo este SQL ou PL/SQL. Podemos, neste caso, utilizar referências como o %TYPE ou %ROWTYPE se necessário, ou mesmo utilizar qualquer tipo de dados escalar ou composto.
IS	Necessário na criação de funções armazenadas
Seção de declaração	Esta sessão é igual as que são feitas após a palavra reservada DECLARE e antes do BEGIN em um bloco anônimo. Podemos criar variáveis, constantes, outros blocos PL/SQL nomeados como procedures, functions etc.
Seção de execução	Contém o bloco PL/SQL que inicia com a cláusula BEGIN e finaliza com END [nome_da_procedure].
Seção de exceções:	Serve para tratarmos as exceções geradas por uma procedure.
[]	Lembrando que tudo que está entre colchetes [ ] é opcional

FONTE: Adaptado de Dionisio (2014)

A seguir temos o exemplo da criação de uma procedure simples, que irá apenas testar os valores de entrada. Neste exemplo, dois valores de entrada são informados.

```
CREATE OR REPLACE PROCEDURE calcula_valor (valor_a IN num
ber, valor_b IN number) IS
valor_saida number;
BEGIN
valor_saida := valor_a + valor_b;
IF (valor_saida > 10) THEN
  DBMS_OUTPUT.put_line (valor_saida);
ELSE DBMS_OUTPUT.put_line ('O valor é menor que o permitido');
 END IF;
END;
```

Vale lembrar que a criação da procedure/função não significa que elas serão executadas automaticamente, somente que a sua criação foi realizada com sucesso, e que não apresenta erros de sintaxe. Como mencionamos, não basta a compilação da procedure, para que ela seja executada, é necessário criar um bloco anônimo e fazer a chamada. Conforme a seguinte sintaxe:

```
BEGIN
     NOME_PRECEDURE;
 END;
```

Com o exemplo apresentado, para executar a procedure é necessário estar entre um Begin e End e os parâmetros. Mas elas também podem ser chamadas dentro de uma função, como veremos adiante.

```
BEGIN
calcula_valor(3,5);
END;
```

Vejamos um exemplo onde a procedure está retornando um valor. Nesta procedure são informados dois valores de entrada (pcd_funcionario e ppr_reajuste) um valor de saída (pvl_novo_salario).

```
 CREATE OR REPLACE PROCEDURE calcular_salario (pcd_funcionario
  IN employees.employee_id%TYPE ,ppr_reajuste     IN NUMBER ,pvl_
novo_salario OUT NUMBER) IS
   BEGIN
     SELECT CASE WHEN NVL(ppr_reajuste,0)=0 THEN
                 salary
          ELSE
              salary + (salary * ppr_reajuste/100)
          END
       INTO pvl_novo_salario
```

```
    FROM employees e
   WHERE e.employee_id = pcd_funcionario;
EXCEPTION
  WHEN OTHERS THEN
     pvl_novo_salario:= -1;
END calcular_salario;
```

Como dito anteriormente, a procedure precisa estar em um bloco anônimo, e, neste caso, ter as variáveis que serão solicitadas.

```
DECLARE
 PCD_FUNCIONARIO NUMBER:= &CD_FUNCIONARIO;
 PPR_REAJUSTE NUMBER:= &PR_REAJUSTE;
 PVL_NOVO_SALARIO NUMBER:= NULL;
BEGIN CALCULAR_SALARIO(PCD_FUNCIONARIO,PPR_REAJUS
TE,PVL_NOVO_SALARIO );
 IF PVL_NOVO_SALARIO = -1 then
  DBMS_OUTPUT.PUT_LINE('Erro ao atualizar' || PVL_NOVO_SA
LARIO);
 ELSE
  DBMS_OUTPUT.PUT_LINE('PVL_NOVO_SALARIO = ' || PVL_
NOVO_SALARIO);
 END IF;
END;
```

Vale ressaltar o resultado em uma procedure que é utilizado para verificar se o código apresenta um erro. Como no exemplo apresentado, caso caio na exception, o valor retornado será -1 (um negativo).

3 *FUNCTIONS*

Como definimos anteriormente, além das procedures temos as funções ou FUNCTION, que de acordo com Fanderuff (2003), são subprogramas que têm por objetivo retornar algum resultado ou valor. Na Unidade 2 do nosso livro didático, conhecemos algumas funções que eram definidas pela própria Oracle, por exemplo, a UPPER, que recebe como parâmetro uma *string* e retorna o valor em maiúsculo. As funções de base podem ser utilizadas nas aplicações como qualquer função predefinida, ou seja, em atribuições a variáveis ou como argumento em comandos SELECT. SINTAXE:

```
CREATE OR REPLACE FUNCTION nome_da_função  (Argumento1 IN
Tipo_de_Dados,
 ArgumentoN IN Tipo_de_Dados)  RETURN  Tipo_de_Dados  IS ou
AS
   declarações
```

```
BEGIN
  bloco PL/SQL
END nome_da_função;
```

Para exemplificar melhor, vamos criar uma função chamada BUSCAR_NOME_CARGO, que a partir do código do funcionário, irá trazer o nome completo do funcionário e a descrição do cargo. Como já vimos anteriormente, no corpo da procedure e function, podemos utilizar os comandos DDL, conforme a necessidade.

```
CREATE OR REPLACE FUNCTION buscar_nome_cargo(pcd_funcionario hr.employees.employee_id%TYPE) RETURN VARCHAR2 IS
  CURSOR c1 IS
      SELECT e.first_name || ' ' || e.last_name nm_funcionario
        , e.job_id     cd_cargo
        , j.job_title  ds_cargo
      FROM hr.employees e
      INNER JOIN hr.jobs j
      ON j.job_id = e.job_id
      WHERE e.employee_id = pcd_funcionario;
  r1    c1%ROWTYPE;
  vtexto VARCHAR2(4000);
BEGIN
  IF c1%ISOPEN THEN
    CLOSE c1;
  END IF;
  OPEN c1;
  FETCH c1 into r1;
  vtexto := r1.nm_funcionario||' ('||r1.ds_cargo||')';
  CLOSE c1;
  RETURN(vtexto);
EXCEPTION
  WHEN OTHERS THEN
    P_insere_erro('Função BUSCAR_NOME_CARGO – Erro: '||SQLERRM);
  END;
```

Observe que no tratamento da exceção, estamos utilizando uma procedure. Este é outro exemplo de utilização de procedure, para armazenar log de execução ou erro. Assim, só fazemos a chamada da procedure, sem a necessidade de escrever todo o comando.

```
CREATE OR REPLACE PROCEDURE P_insere_erro(p_erro in varchar2)
IS
BEGIN
  INSERT INTO controla_erro VALUES(p_erro, sysdate);
END;
```

Diferente da chamada da procedure, a chamada da função pode acontecer dentro de um select, ou em uma mensagem.

Retorno na dual: **SELECT** buscar_nome_cargo(100) **FROM DUAL**;
Retorno em select:
SELECT buscar_nome_cargo(e.employee_id), e.salary
FROM jobs j,
employees e
WHERE e.job_id = j.job_id

4 APAGANDO OBJETOS

Algumas vezes precisamos aparar um objeto no banco, seja por não ser mais útil, ou por ter sido criado incorretamente. Para realizar essa operação e apagar um objeto criado no banco de dados, utilizamos a instrução DROP, conforme as seguintes sintaxes:

- DROP FUNCTION [SCHEMA.]nome_funcao;
- DROP PROCEDURE [SCHEMA.]nome_procedimento;

Mas lembre-se de que uma vez executado esse comando, ele não pode ser desfeito. Então é necessário muita atenção na hora de utilizar.
Exemplo: DROP FUNCTION f_nome_funcionario;

LEITURA COMPLEMENTAR

TRIGGER ORACLE (BÁSICO)

Roberto Fernandes Sobrinho

Este artigo relata de forma clara e objetiva as funcionalidades básicas de Triggers utilizadas em banco de dados Oracle.

1 Introdução

Em um banco de dados temos 4 ações possíveis (Insert, Update, Delete e Select), temos 3 dessas ações como possíveis modificadoras do banco de dados (select não modifica o banco de dados). Os bancos de dados preveem que caso haja uma ação de modificação podemos ter ações complementares vinculadas a mesma, que é o que chamamos de Trigger.

Trigger como a tradução se faz, são gatilhos, o gatilho ocorre toda vez que uma ação ocorre em um banco de dados. O Trigger é consequência de uma ação, portanto não é o fim e sim algo que ocorre em função de uma ação.

Os Triggers podem ser do tipo ROW LEVEL (Linha) ou STATEMENT (Tabela). Além disso, os Triggers podem ocorrer (BEFORE) antes ou (AFTER) depois que a ação tenha ocorrido, e por fim pode ocorrer para as ações de INSERT, UPDATE e DELETE em uma tabela.

2 Tipos de TRIGGERS

Trigger ROW LEVEL

Os Triggers do tipo ROW LEVEL podem ser usados sempre que precisarmos que um Trigger trate de valores em uma transação, e por sua vez são disparados a cada ocorrência de uma transação sobre uma tabela. Se um UPDATE atualizar, por exemplo, 1000 linhas em uma tabela que possua um Trigger de Update do tipo row level, serão disparadas 1000 vezes.

Os Triggers do tipo row level são utilizadas para operações como:

- Gravação de LOGS de auditoria de uma aplicação;
- Verificação de dados (Consistência);
- Implementação de integridade referencial;

Trigger STATEMENT

Os Triggers do tipo STATEMENT tem a finalidade de tratar a execução de ações sobre tabelas independentemente de quantas linhas forem afetadas. Através deste tipo de Trigger podemos registrar a execução de comandos INSERT, UPDATE e DELETE contra tabelas que tenham Triggers contemplando essas ações. Caso um comando UPDATE atualize 1000 linhas, um Trigger deste tipo apenas dispararia 1 única vez. Este tipo de Trigger não pode referenciar qualquer valor contido em uma coluna da tabela. Isso ocorre porque se o mesmo dispara uma única vez.

Este tipo de Trigger funciona nos casos de registro de transações ocorridas, independentemente do número de linhas afetadas.

Exemplo:

```
CREATE OR REPLACE TRIGGER trg_aud_trn
BEFORE INSERT OR DELETE OR UPDATE
ON transportador
REFERENCING NEW AS NEW OLD AS OLD
BEGIN
IF TO_NUMBER (TO_CHAR (SYSDATE, "hh24")) NOT BETWEEN 9 AND 18
THEN
raise_application_error(-20001,"Operação não pode ser executada fora do horário de expediente.");
END IF;
```

```
END;
/
```

COLUMN Trigger: Triggers de Coluna são disparados sempre que a determinada coluna relacionada no Trigger sofrer a ação ligada ao mesmo. Com isso se a ação ocorrer em outras colunas que não aquelas associadas ao Trigger, o Trigger não será disparado. Isso evita de certa forma que um Trigger seja disparado se a ação não ocorra, nada será disparado. Isso pode ser útil em casos de auditoria de mudanças em determinadas colunas, por exemplo:

Toda vez que um salário for alterado na tabela de empregados um Trigger gravará em uma tabela chamada auditoria_empregado um registro contendo o código do empregado e os salários anteriores e atuais.

TABLE Trigger: Trigger de tabela ocorrem independentemente das colunas afetadas pela ação. Isso quer dizer que se uma ação ocorre, não importa qual coluna seja afetada, a ação que o Trigger cobre não está ligado a nenhuma coluna. Estes Triggers podem ser úteis em ações como o exemplo a seguir;

Toda vez que um pedido for alterado (não importa a coluna) gravaremos em uma tabela chamada auditoria_pedido todas as colunas afetadas pelo Trigger.

3 Momento (Antes ou Depois)

Before – (Antes): Os Triggers do tipo BEFORE como podemos deduzir, disparam antes que a ação ocorra. Isso leva a entender que antes que uma ação de banco de dados ocorra o Trigger será disparado, o que pode fazer com que a ação nem venha a ocorrer. Um Trigger pode impedir que uma ação venha a ocorrer, portanto podemos usar um Trigger deste tipo em situações como, por exemplo:

- Validação de dados;
- Carregamento de dados obrigatórios (datas, usuários, etc.);
- Impedimento de ações em horários não previstos;

Exemplo:

```
CREATE OR REPLACE TRIGGER trg_pedidos_valor
BEFORE INSERT OR DELETE OR UPDATE
ON pedidos
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
IF :NEW.PEDI_VL_BRUT_CALC < 10000 THEN
THEN
raise_application_error(-20001,"O valor do pedido deve ser inferior a
10.000,00");
END IF;
END;
/
```

After – (Depois): Os Triggers do tipo AFTER ocorrem depois que a ação tenha ocorrido, ou seja eles são disparados depois, com isso NÃO podemos com esses tipos de Triggers fazer o que fazemos com Triggers do tipo BEFORE. Aqui a ação já ocorreu então o que podemos fazer com Triggers deste tipo é a auditoria.

Exemplo:

```
CREATE OR REPLACE TRIGGER trg_salario_aud
AFTER UPDATE
ON empregados
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
INSERT INTO log_salario
(codigo, salario_anterior, salario_atual,
data_alteracao, usuario )
VALUES (:NEW.codigo, :OLD.salario_anterior,
:NEW.salario_atual, SYSDATE, USER);
END;
/
```

4 Modificadores OLD e NEW:

Podemos nos casos de Triggers de linha, fazer referência a valores contidos nas colunas e com isso podemos querer saber os valores antes da alteração e depois dos valores efetivamente alterados. Isso vale na ação de UPDATE, nos casos de INSERT e DELETE os valores de OLD (INSERT) e NEW (DELETE) são nulos. Estes modificadores podem ser usados APENAS em TRIGGERS. Não podemos usá-los em procedures, functions ou packages. Os valores são referenciados da seguinte forma :OLD.nomecoluna e :NEW.nomecoluna.

Não importa se o Trigger for BEFORE ou AFTER os modificadores OLD e NEW não são afetados. Exemplo:

```
CREATE OR REPLACE TRIGGER trg_salario_aud
AFTER UPDATE
ON empregados
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
IF :NEW.SALARIO < 550 THEN
:NEW.SALARIO := 550;
END IF;
END;
```

5 Cláusula WHEN:

Caso o Trigger tenha alguma condição para ser executado, podemos incluir uma cláusula chamada WHEN. Nesta colocamos as condições que o Trigger irá disparar. Caso precisemos tratar o valor de alguma coluna, usamos os modificadores OLD e NEW, mas nessa clausula não colocaremos os : na frente, pois nesse caso ocorrerá erro. Exemplo:

```
CREATE OR REPLACE TRIGGER trg_salario_aud
AFTER UPDATE
ON empregados
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
WHEN (NEW.salario < 550)
BEGIN
:NEW.salario := 550;
END;
/
```

No exemplo acima o Trigger somente será executado (BEGIN... END) se a condição WHEN vier a ocorrer.

6 Operadores, INSERTING, UPDATING E DELETING:

Podemos criar Triggers para serem disparados para várias ações de banco de dados. Dessa forma como podemos diferenciar uma ação de insert de outra de delete ou ainda update. Estes modificadores podem ser feitos APENAS dentro de Triggers, portanto se usarmos esses operadores em procedures, functions, Packages ou mesmo PL/SQL anônimos os mesmos ocasionarão erros de compilação. Exemplo:

```
CREATE OR REPLACE TRIGGER TRG_SALARIO_AUD
AFTER
INSERT OR DELETE OR UPDATE
ON EMPREGADOS
FOR EACH ROW
BEGIN
IF INSERTING THEN
INSERT INTO LOG_SALARIO (CODIGO,
SALARIO_ANTERIOR,
SALARIO_ATUAL,
DATA_ALTERACAO,
USUARIO)
VALUES (:NEW.CODIGO,
NULL, :NEW.SALARIO_ATUAL,
SYSDATE, USER);
ELSIF UPDATING THEN
INSERT INTO LOG_SALARIO (CODIGO,
SALARIO_ANTERIOR,
SALARIO_ATUAL,
DATA_ALTERACAO,
USUARIO)
VALUES (:NEW.CODIGO,
:OLD.SALARIO_ATUAL, :NEW.SALARIO_ATUAL,
SYSDATE, USER);
ELSIF DELETING THEN
```

```
INSERT INTO LOG_SALARIO (CODIGO,
SALARIO_ANTERIOR,
SALARIO_ATUAL,
DATA_ALTERACAO,
USUARIO)
VALUES (:NEW.CODIGO,
:OLD.SALARIO_ATUAL, NULL,
SYSDATE, USER);
END IF;
END;
/
```

7 Comando INSTEAD OF:

Triggers deste tipo foram implementadas a partir da versão 9i e tem a finalidade de permitir que havendo ações de modificação sobre visões, que os comandos possam ser realizados nas tabelas associadas a essas visões. Exemplo:

```
CREATE OR REPLACE TRIGGER TRG_NOVO_FUNC
INSTEAD OF INSERT ON EMPREGADOS
FOR EACH ROW
BEGIN
INSERT INTO empregados
(codigo, nome, salario)
VALUES (:NEW.codigo, :NEW.nome, :NEW.salario);
END;
```

8 Restrições ao uso de Triggers

Não podemos realizar os comandos COMMIT, ROLLBACK e SAVE-POINT em um Trigger, mesmo que seja uma procedure executada em um Trigger.

Não podemos fazer select na mesma tabela que sofre a ação de um Trigger, pois isso pode provocar um erro chamado MUTANT TABLE. Mesmo porque se quisermos saber o valor de uma coluna do registro que está sendo tratado em um Trigger basta colarmos :new.nomecoluna ou :old.nomecoluna para termos respectivamente os valores atuais e anteriores a alteração.

Triggers tornam as operações mais lentas, isso ocorre principalmente em casos de Triggers de linha.

FONTE: <https://bit.ly/2VuMZ4e>. Acesso em: 31 mar. 2020.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- O PL/SQL permite que se criem programas inteiros em sua estrutura.
- Durante a programação em banco de dados, podemos utilizar as instruções de controle condicional.
- Podemos obter muitas vantagens na utilização do ambiente PL/SQL e seus tipos de blocos de comandos.
- Devemos utilizar e tratar erros e exceções, durante o desenvolvimento de procedures e functions.

Ficou alguma dúvida? Construímos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

AUTOATIVIDADE

Seguiremos com a nossa base de dados utilizada nos tópicos anteriores.

Tabela Cargo

CdCargo	DescCargo	VlrSalario
C1	Aux.Vendas	350,00
C2	Vigia	400,00
C3	Vendedor	800,00
C4	Aux. Cobrança	250,00
C5	Gerente	1000,00
C6	Diretor	2500,00

Tabela Depto

CdDepto	DescDepto	RamalTel
D1	Assist.Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688

Tabela Funcionário

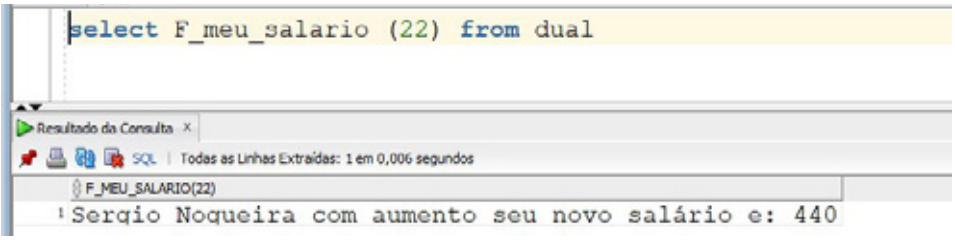
NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
195	Marta Silveira	05/01/2002	F	C1	D5
139	Ana Luiza	12/01/2003	F	C4	D6
123	Pedro Sergio	29/06/2003	M	C7	D3
148	Larissa Silva	01/06/2002	F	C4	D6
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

1 Transforme o bloco anônimo criado no exercício anterior em uma procedure com o nome Calcula_valor_anual.

2 Com base na procedure criada anteriormente, crie outra chamada calcula_valor_anual_2 para que grave os erros ocorridos no exception sejam gravados em uma tabela de log, chamada controla_erro, que deverá ter as seguintes colunas: (dsc_erro varchar2(100), Nome_tabela varchar2(30), data_erro date).

3 Vamos melhorar o exercício anterior, e ao invés de fazermos insert na tabela de log que criamos, vamos criar uma procedure com o nome P_grava_log que irá receber as colunas da tabela como parâmetros de entrada. Feito isso, vamos criar a procedure calcula_valor_anual_3, que irá chamar a P_grava_log , onde teremos uma procedure chamando outra procedure.

4 Crie uma função chamada F_meu_salario que receba como parâmetro o seu número de registro e retorne o seu nome e o seu novo salário com 10% de aumento. A mensagem de retorno da função deve ser, conforme o apresentado na execução:



5 Altere a função chamada ganho_anual, e inclua a procedure desenvolvida na atividade 2 chamada Grava_log.

REFERÊNCIAS

ALÉSSIO, S. C. **Banco de dados avançado**. Indaial: UNIASSELVI, 2015.

ALVES, E. S.; JÚNIOR, J. C. C. **Tipos de Joins**: Padrões ANSI92 (SQL92) e SQL89. 2015. Disponível em: https://bit.ly/2RCAaUB. Acesso em: 15 out. 2109.

ALVES, W. P. **Banco de dados**. São Paulo: Érica, 2014.

AUDY, J. N.; ANDRADE, G. de; CIDRAL, A. **Fundamentos de sistemas de informação**. Porto Alegre: Bookman, 2007.

BALIEIRO, R. **Banco de dados**. Rio de Janeiro: SESES, 2015. 168 p.

BARBOZA, F. F. M. **Modelagem e desenvolvimento de banco de dados** [recurso eletrônico] Porto Alegre: SAGAH, 2018.

BRUMM, B. **Beginning Oracle SQL for Oracle Database 18c**: From Novice to Professional. Berkeley: Apress, Agosto 2019. DOI 10.1007/978-1-4842-4430-2. XXVI, 431 p. ISBN 978-1-4842-4430-2.

CABEÇA, A. G. **Análise de mutantes em aplicações SQL de banco de dados**. Campinas, SP: 2009.

CAMARGO, W. B. de. **Conceitos e criação de views no SQL Server 2011**. Disponível em: https://bit.ly/2VwGQ7C. Acesso em: 5 jan. 2020.

COSTA, P. A. **Base pessoal de conhecimento em engenharia de software, programação e informática em geral**. 2012. Disponível em: http://www.pauloacosta.com/tag/banco-de-dados/. Acesso em: 8 mar. 2020.

COSTA, T. **Merge**. 2015. Disponível em: https://bit.ly/2RwWuyP. Acesso em: 15 out. 2019.

Devmedia. **Álgebra relacional**. 2008. Disponível em: https://bit.ly/2XHdiqu. Acesso em: 26 jan. 2020.

DIONISIO, E. J. **Trabalhando com estruturas de controle no Oracle**. 2015. Disponível em: https://bit.ly/2XzYGcI. Acesso em: 6 jan. 2019.

FANDERUFF, D. **Dominando o Oracle 9i**: modelagem e desenvolvimento. São Paulo: Pearson Educations do Brasil, 2003.

GOMES, E. H. **Sistema gerenciador de banco de dados**. 2012. Disponível em: http://ehgomes.com.br/disciplinas/bdd/sgbd.php. Acesso em: 8 mar. 2020.

GONÇALVES, E. C. **Tratamento de exceções de sistema na linguagem PQ/SQL**. Rio de Janeiro: DevMedia, 2012. Disponível em: https://bit.ly/3cjsHBG. Acesso em: 6 jan. 2020.

GUPTA, S. K., **Advanced Oracle PL/SQL Developer's Guide – Second Edition**, 2ª ed. Birmingham: Packt Publishing, Limited, 2016.

HEUSER, C. A. **Projeto de banco de dados**. Porto Alegre: Instituto de informática da UFRGS, 2001.

HEUSER, C. A. **Projeto de banco de dados**. Rio Grande do Sul. Bookman editora, 2009.

JAYAPALAN, L. *et al*. **Oracle Database PL/SQL Language Reference**: E96448-02 19c. EUA: Oracle, Janeiro 2019. Disponível em: https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/index.html. Acesso em: 6 jan. 2020.

JÚNIOR GALVÃO, P. **Conhecendo a álgebra relacional e seus operadores**. 2016. Disponível em: https://bit.ly/2Vq8IdC. Acesso em: 26 jan. 2020.

KERYAN A. **Analysus bd Desing of Data Systems**, 2015. Disponível em: https://en.ppt-online.org/91574. Acesso: 7 jan. 2020.

LEHMKUHL, D.; EGER, D. R. **Princípios de banco de dados**. Indaial: Uniasselvi, 2013.

LESSA, J. R. **Teoria dos conjuntos**. (2018). Disponível em: https://www.infoescola.com/matematica/teoria-dos-conjuntos/. Acesso em: 26 jan. 2020.

MACHADO, F. N. R. **Projeto e implementação de banco de dados**. São Paulo: Érica, 2014.

MACORATTI, J. C. **SQL – Álgebra relacional – operações fundamentais – conceitos básicos**. 2013. Disponível em: https://bit.ly/2V5IbmV. Acesso em: 26 jan. 2020.

MCLAUGHLIN, M. **Oracle Database 12c PL/SQL Programming**. [S.l.]: McGraw-Hill, 2014.

MORAES, E. **Apostila-Banco-de-Dados**, 2015. Disponível em: https://pt.scribd.com/document/280354088/Apostila-Banco-de-Dados. Acesso em: 6 jan. 2020.

MORAIS, R. S. de. **Programando em Oracle 9i PL/SQL**. Brasil: [*s. n.*], 2001. Disponível em: https://bit.ly/3bcfiLm. Acesso em: 5 jan. 2020.

NUITJEN, A.; WIDLAKE, M.; HELSKYAHO, H.; TIERNEY, B.; NANDA, A. **Real World SQL and PL/SQL**: Advice from the Experts. New York: McGraw-Hill, Agosto 2016. 512 p. *E-book*.

ORACLE. SQL Workseet. **Oracle Live SQL**, 2019. Disponivel em: https://livesql.oracle.com/. Acesso em: 15 nov. 2019.

PASCUTTI, M. C. D. **Administrador de banco de dados**. 2012. Disponível em: https://bit.ly/2Vxs9Bc. Acesso em: 26 jan. 2020.

PLATAFORMA DEVMEDIA. **Artigo SQL Magazine 53 – Modelagem Relacional**. 2008. Disponível em: https://bit.ly/2XAA2bZ. Acesso em: 8 mar. 2020.

PLATAFORMA DEVMEDIA. **Banco de dados relacional**. 2011. Disponível em: https://bit.ly/2V72pwv. Acesso em: 8 mar. 2020.

PLATAFORMA DEVMEDIA. **Modelo Entidade Relacionamento (MER) e Modelo Entidade-Relacionamento (DER)**. 2014. Disponível em: https://bit.ly/2UKqzMe. Acesso em: 8 mar. 2020.

PORTAL EDUCAÇÃO. **SQL Alias**. Disponivel em: https://bit.ly/34F5xD1. Acesso em: 4 jan. 2020.

PRICE, J. **Oracle Database 11g SQL**: Domine SQL e PL/SQL no banco de dados Oracle. São Paulo: Bookman, 2009.

RAMARKRISHNAN, J. G. **Sistemas de gerenciamento de banco de dados**. 3. ed. Porto Alegre: AMGH, 2011.

SANCHES, A. R. **Disciplina**: fundamentos de armazenamento e manipulação de dados. 2005. Disponível em: https://bit.ly/2K6MAzD. Acesso em: 8 mar. 2019.

SANTOS, M. A. da S. **Física e a evolução dos meios de armazenamento de informações**. s.d. Disponível em: https://bit.ly/2Vavdoa. Acesso em: 8 mar. 2020.

SOUZA, O. **Edgar Frank Codd and the relational database**: a contribution to the History of Computing. 2015. Dissertação (Mestrado em História da Ciência) - Pontifícia Universidade Católica de São Paulo, São Paulo, 2015.

WILLIAM, M. **Programando com cursores PL/SQL**, 2009. Disponível em: https://bit.ly/2z1Kn6v. Acesso em: 7 jan. 2020.