



## BANCO DE DADOS II

Professor Me. William Roberto Pelissari

Professor Esp. Carlos Danilo Luz

Professor Esp. Jeferson Kaiser

GRADUAÇÃO

**Unicesumar**

**UNICESUMAR**

Av. Guedner, 1610 - Jardim Aclimação  
Cep 87050-900 - MARINGÁ - PARANÁ  
unicesumar.edu.br  
44 3027.6360

**UNICESUMAR EDUCAÇÃO A DISTÂNCIA**

NEAD - Núcleo de Educação a Distância  
Bloco 4 - MARINGÁ - PARANÁ  
unicesumar.edu.br  
0800 600 6360

as imagens utilizadas neste  
livro foram obtidas a partir  
do site SHUTTERSTOCK.COM

**FICHA CATALOGRÁFICA**

C397 **CENTRO UNIVERSITÁRIO DE MARINGÁ.** Núcleo de Educação a Distância; **PELISSARI**, William Roberto; **LUZ**, Carlos Danilo; **KAISER**, Jeferson.

**Banco de Dados II.** William Roberto Pelissari; Carlos Danilo Luz; Jeferson Kaiser.

Maringá-Pr.: UniCesumar, 2018. Reimpresso em 2021.  
171 p.

"Graduação - EaD".

1. Banco. 2. Dados. 3. EaD. I. Título.

ISBN 978-85-459-0707-7

CDD - 22 ed. 005.75  
CIP - NBR 12899 - AACR/2

Ficha catalográfica elaborada pelo bibliotecário  
João Vivaldo de Souza - CRB-8 - 6828  
Impresso por:

**Reitor**

Wilson de Matos Silva

**Vice-Reitor**

Wilson de Matos Silva Filho

**Pró-Reitor Executivo de EAD**

William Victor Kendrick de Matos Silva

**Pró-Reitor de Ensino de EAD**

Janes Fidélis Tomelin

**Presidente da Mantenedora**

Cláudio Ferdinandi

**NEAD - Núcleo de Educação a Distância****Diretoria Executiva**

Chrystiano Mincoff

James Prestes

Tiago Stachon

**Diretoria de Graduação e Pós-graduação**

Kátia Coelho

**Diretoria de Permanência**

Leonardo Spaine

**Diretoria de Design Educacional**

Débora Leite

**Head de Produção de Conteúdos**

Celso Luiz Braga de Souza Filho

**Head de Curadoria e Inovação**

Jorge Luiz Vargas Prudencio de Barros Pires

**Gerência de Produção de Conteúdo**

Diogo Ribeiro Garcia

**Gerência de Projetos Especiais**

Daniel Fuverki Hey

**Gerência de Processos Acadêmicos**

Taessa Penha Shiraishi Vieira

**Gerência de Curadoria**

Giovana Costa Alfredo

**Supervisão do Núcleo de Produção****de Materiais**

Nádila Toledo

**Supervisão Operacional de Ensino**

Luiz Arthur Sanglard

**Coordenador de Conteúdo**

Fabiana de Lima

**Qualidade Editorial e Textual**

Daniel F. Hey, Hellyery Agda

**Design Educacional**

Isabela Ventura, Ana Claudia Salvadego, Agnaldo Ventura

**Iconografia**

Isabela Soares Silva

**Projeto Gráfico**

Jaime de Marchi Junior

José Jhonnny Coelho

**Arte Capa**

Arthur Cantareli Silva

**Editoração**

Robson Yuiti Saito

**Revisão Textual**

Pedro Afonso Barth

**Ilustração**

Bruno Cesar Pardinho



Professor  
**Wilson de Matos Silva**  
Reitor

Viver e trabalhar em uma sociedade global é um grande desafio para todos os cidadãos. A busca por tecnologia, informação, conhecimento de qualidade, novas habilidades para liderança e solução de problemas com eficiência tornou-se uma questão de sobrevivência no mundo do trabalho.

Cada um de nós tem uma grande responsabilidade: as escolhas que fizermos por nós e pelos nossos farão grande diferença no futuro.

Com essa visão, o Centro Universitário Cesumar assume o compromisso de democratizar o conhecimento por meio de alta tecnologia e contribuir para o futuro dos brasileiros.

No cumprimento de sua missão – “promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária” –, o Centro Universitário Cesumar busca a integração do ensino-pesquisa-extensão com as demandas institucionais e sociais; a realização de uma prática acadêmica que contribua para o desenvolvimento da consciência social e política e, por fim, a democratização do conhecimento acadêmico com a articulação e a integração com a sociedade.

Diante disso, o Centro Universitário Cesumar almeja ser reconhecido como uma instituição universitária de referência regional e nacional pela qualidade e compromisso do corpo docente; aquisição de competências institucionais para o desenvolvimento de linhas de pesquisa; consolidação da extensão universitária; qualidade da oferta dos ensinos presencial e a distância; bem-estar e satisfação da comunidade interna; qualidade da gestão acadêmica e administrativa; compromisso social de inclusão; processos de cooperação e parceria com o mundo do trabalho, como também pelo compromisso e relacionamento permanente com os egressos, incentivando a educação continuada.





## Janes Fidélis Tomelin

Pró-Reitor de Ensino de EaD

## Kátia Solange Coelho

Diretoria de Graduação e Pós

## Débora do Nascimento Leite

Diretoria de Design Educacional

## Leonardo Spaine

Diretoria de Permanência

Seja bem-vindo(a), caro(a) acadêmico(a)! Você está iniciando um processo de transformação, pois quando investimos em nossa formação, seja ela pessoal ou profissional, nos transformamos e, consequentemente, transformamos também a sociedade na qual estamos inseridos. De que forma o fazemos? Criando oportunidades e/ou estabelecendo mudanças capazes de alcançar um nível de desenvolvimento compatível com os desafios que surgem no mundo contemporâneo.

O Centro Universitário Cesumar mediante o Núcleo de Educação a Distância, o(a) acompanhará durante todo este processo, pois conforme Freire (1996): “Os homens se educam juntos, na transformação do mundo”.

Os materiais produzidos oferecem linguagem dialógica e encontram-se integrados à proposta pedagógica, contribuindo no processo educacional, complementando sua formação profissional, desenvolvendo competências e habilidades, e aplicando conceitos teóricos em situação de realidade, de maneira a inseri-lo no mercado de trabalho. Ou seja, estes materiais têm como principal objetivo “provocar uma aproximação entre você e o conteúdo”, desta forma possibilita o desenvolvimento da autonomia em busca dos conhecimentos necessários para a sua formação pessoal e profissional.

Portanto, nossa distância nesse processo de crescimento e construção do conhecimento deve ser apenas geográfica. Utilize os diversos recursos pedagógicos que o Centro Universitário Cesumar lhe possibilita. Ou seja, acesse regularmente o AVA – Ambiente Virtual de Aprendizagem, interaja nos fóruns e enquetes, assista às aulas ao vivo e participe das discussões. Além disso, lembre-se que existe uma equipe de professores e tutores que se encontra disponível para sanar suas dúvidas e auxiliá-lo(a) em seu processo de aprendizagem, possibilitando-lhe trilhar com tranquilidade e segurança sua trajetória acadêmica.

### **Professor Esp. Jeferson Kaiser**

Especialista em Administração de Banco de Dados Oracle/DB2 pelo Centro de Ensino Superior Cesumar (UNICESUMAR/2015) e em Contabilidade Gerencial Controladoria e Auditoria pela Faculdade de Jandaia do Sul (FAFIJAN/2013). É graduado em Ciência da Computação pela Faculdade de Filosofia Ciências e Letras de Mandaguari (FAFIMAN/2012). Atualmente é professor mediador do Centro Universitário de Maringá e analista de sistemas jr. da empresa Matera Systems. Tem experiência na área de Ciência da Computação, com ênfase em Arquitetura de Sistemas de Computação.

Para informações mais detalhadas sobre sua atuação profissional, pesquisas e publicações, acesse seu currículo, disponível no endereço a seguir:

<<http://lattes.cnpq.br/1926432986663027>>.

### **Professor Me. William Roberto Pelissari**

Mestre em Desenvolvimento de Tecnologia pelo Instituto de Tecnologia para o Desenvolvimento (Lactec/2014). Especialista em Aplicação para Internet e Dispositivos Móveis pela Universidade Paranaense (UNIPAR/2014). Especialista em Administração de Produção e Logística pela Faculdade Estadual de Educação Ciências e Letras de Paranavaí (FAFIPA/2010). Especialista em Engenharia de Software pela Universidade Norte do Paraná (UNOPAR/1997). Graduado em Tecnologia em Processamento de Dados pela Unopar (1995). Graduação em andamento em Processos Gerenciais pela Unipar. Atualmente atua na Gestão Educacional da Virtual Age by TOTVS. É professor e coordenador da pós-graduação da Unipar e professor no Ensino a Distância (EAD) do Centro Universitário Cesumar (Unicesumar). Tem experiência na área de Ciência da Computação com ênfase em Projetos de Software, Engenharia de Software, Análise e Desenvolvimento de Software e Gerência, Governança e Infraestrutura de TI.

Para informações mais detalhadas sobre sua atuação profissional, pesquisas e publicações, acesse seu currículo, disponível no endereço a seguir:

<<http://lattes.cnpq.br/1381263791825712>>.

### **Professor Esp. Carlos Danilo Luz**

Especialista em Educação a Distância (EAD) e as Tecnologias Educacionais pelo Centro Universitário Cesumar (UniCesumar/2016). Graduado em Redes de Computadores pelo UniCesumar (2009). Especializações em andamento de Gestão Estratégica de Pessoas e Docência no Ensino Superior pela mesma instituição. Atualmente é professor da Secretaria de Educação do Estado do Paraná. Experiência de 9 anos em Desenvolvimento de Sistemas Web e Projetos.

Para informações mais detalhadas sobre sua atuação profissional, pesquisas e publicações, acesse seu currículo, disponível no endereço a seguir:

<<http://lattes.cnpq.br/7063667454769099>>.

# BANCO DE DADOS II

## SEJA BEM-VINDO(A)!

Caro (a) aluno (a), seja bem-vindo (a)!

Este material visa permitir que você possa compreender diversos conceitos avançados em Banco de Dados. Nas unidades serão abordadas desde funcionalidades específicas até controles de permissões. Sendo assim, um Administrador de Banco de Dados deve ter o conhecimento relacionado à estrutura, e também conhecimento de comandos para a manipulação de um Sistema de Gerenciamento de Banco de Dados (SGBD).

Os conceitos aplicados neste material têm como base o Banco de Dados Oracle e MySQL, por serem considerados grandes líderes de mercado neste segmento.

As unidades, que serão abordadas neste livro, têm como intuito preparar você aluno(a) para se tornar um profissional apto a manipular as informações em um SGBD. Será apresentada na Unidade I a linguagem Structured Query Language (SQL), a mesma é considerada como a linguagem padrão dos SGBDs que tem como base o modelo de dados relacional, além de funcionalidades que irão proporcionar a você, aluno(a), trabalhar com o banco de dados.

Na Unidade II, vamos conhecer melhor os comandos SQL utilizando o SGBD MySQL. Sendo assim, iremos trabalhar com a manipulação de funções SQL distribuídas da seguinte forma: criação de uma base de dados, criação e alteração de tabelas, inserção e alteração de dados, realização de consultas, além de estudarmos sobre controle de dados e transações.

Como sequência a unidade III aborda comandos avançados em SQL, assim explorando melhor as funcionalidades de um banco de dados. Por sua vez, a unidade IV foi elaborada pensando em proporcionar a você aluno(a) conhecimentos sobre o Procedural Language/Structured Query Language (PL/SQL), utilizando o SGBD Oracle, vamos também nos aprofundar nos conceitos e exemplos de Procedures, Funções e Pacotes.

Por fim, na Unidade V iremos trabalhar com Triggers, na qual vamos abrir um grande leque de situações a serem trabalhadas em um Banco de Dados. Além disso, devemos focar nossos olhos na segurança das informações, sabendo que elas são consideradas a alma de uma empresa. Com base nessa informação, torna-se necessário termos um controle de acesso às informações de nosso Banco de Dados.

Desejamos um bom proveito do material e ótimo estudo!



# SUMÁRIO

## UNIDADE I

### CRIANDO UM BANCO DE DADOS

15	Introdução
16	Schema
20	Tipos de Dados
22	Criação e Alteração de Tabelas
25	Chave Primária
28	Populando as Tabelas Criadas
31	Comando Describe
33	Considerações Finais
37	Referências
38	Gabarito

## UNIDADE II

### A LINGUAGEM SQL

41	Introdução
42	História da SQL
44	DQL - Linguagem de Consulta de Dados
49	DML - Linguagem de Manipulação de Dados
51	DDL - Linguagem de Definição de Dados
56	DCL - Linguagem de Transação de Dados
59	DTL - Linguagem de Transação de Dados



# SUMÁRIO

---

61 Considerações Finais

---

65 Referências

---

66 Gabarito

## UNIDADE III

### **MANIPULAÇÃO DE DADOS**

---

69 Introdução

---

70 Extrair Dados de uma Tabela

---

75 Agrupando a Exibição dos Dados

---

76 Ordenação na Exibição dos Dados

---

77 Criando Consultas com Filtros Específicos

---

79 Valores Nulos

---

79 Rename

---

81 Consulta Utilizando Mais de uma Tabela

---

82 Consultas com Subqueries

---

83 Teste de Relações Vazias

---

84 Alterando os Dados com Update

---

86 Removendo os Dados com Delete

---

87 Rollback e Commit

---

88 Truncate e Drop



# SUMÁRIO

---

89 Considerações Finais

---

93 Referências

---

94 Gabarito

## UNIDADE IV

### **PROGRAMAÇÃO EM SQL**

---

99 Introdução

---

100 Visão Geral Sobre PL/SQL

---

104 Procedures

---

119 Functions

---

124 Packages

---

130 Considerações Finais

---

137 Referências

---

138 Gabarito



# SUMÁRIO

## UNIDADE V

### **CONTROLANDO ACESSOS**

---

143 Introdução

---

144 Triggers (Gatilhos)

---

149 Segurança do Banco

---

150 Controle de Acesso ao Usuário

---

156 Criar e Acessar Vínculos de Banco de Dados

---

158 Gerenciando Senhas e Recursos

---

161 Gerenciando Usuários

---

162 Auditoria

---

164 Considerações Finais

---

169 Referências

---

170 Gabarito

---

**171 CONCLUSÃO**



# CRIANDO UM BANCO DE DADOS

## Objetivos de Aprendizagem

- Conhecer como se cria um Banco de Dados.
- Compreender a Criação e alteração de tabelas na prática.
- Verificar quais tipos de dados utilizar nos campos das tabelas.
- Conhecer chave primária na prática.
- Inserir dados nas tabelas.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- *Schema*
- Tipos de Dados
- Criação e Alteração de Tabelas
- Chave Primária
- Populando as Tabelas Criadas
- Comando *Describe*



## INTRODUÇÃO

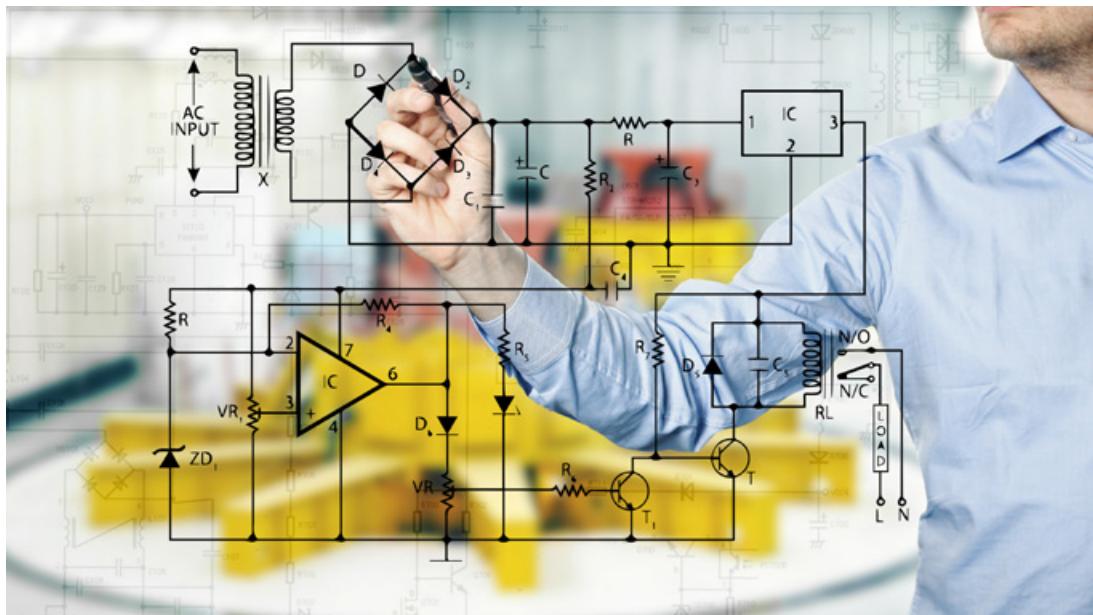
Olá, caro (a) aluno(a)! Nesta unidade, iremos abordar a criação de um Banco de Dados no MySQL, ilustrando o passo a passo, para a criação de um Banco de Dados de fato, criação, com a alteração e remoção das tabelas.

Iremos também popular as tabelas criadas. Nossa intenção é fazer com que você acadêmico (a) verifique na prática a manipulação desde a criação de um schema, até popular dados nas tabelas. Alertamos que neste momento não é necessário a preocupação com a consulta de dados, pois esse assunto será abordado na próxima unidade. Vamos praticar, pois com a prática, você irá entender melhor todos os conceitos vistos até o momento.

Logo quando falamos em Banco de Dados, a primeira coisa que nos vêm à cabeça é a linguagem *Structured Query Language* (SQL). Essa linguagem é muito madura e instável no mercado de Banco de Dados. Fique muito atento com os detalhes dos exemplos de comandos trabalhados nesta unidade, pois um pequeno detalhe fará com que seu comando não execute corretamente na ferramenta que iremos utilizar. Assim você será impossibilitado de concluir todos os comandos.

Nossa ideia para este livro, é que se você executar todos os passos das unidades, será criado um cadastro de pessoa completo, contendo dados para que seja possível você colocar em prática todos os assuntos abordados no livro.

Ótimo estudo!



## SCHEMA

Antes do SQL-92 não existia o conceito de *schema*, em que todas as tabelas e demais arquivos do Banco de Dados eram criados dentro de um mesmo ambiente, não existindo assim um agrupamento de tabelas. Um *schema* é representado por uma coleção de vários objetos de um ou mais usuários de Banco de Dados, como exemplo: tabelas, sequências, índices etc. São associados a um Banco de Dados em razão de vários esquemas para um Banco de Dados, facilitando a administração dos objetos e dos dados.

O comando a seguir é responsável pela criação de um *schema*:

```
CREATE SCHEMA UniCesumar;
```

Em que UniCesumar é o nome do *schema* a ser criado pelo comando. Uma das principais vantagens da utilização de *schemas* em Banco de Dados são as permissões que se pode atribuir e revogar ao usuário, pois podemos dessa maneira ter vários *schemas* e vários usuários em um Banco de dados. Entretanto, determinados usuários têm acesso a somente um *schema* no Banco de Dados.

## BANCO DE DADOS MYSQL

O MySQL é um Sistema de Gerenciamento de Banco de Dados (SGBD) que utiliza a linguagem SQL como interface com o usuário. É também um dos Bancos de Dados mais populares para a Web. É rápido, confiável e fácil de utilizar.

Em todo o nosso desenvolvimento vamos utilizar o MySQL *Workbench* que é a ferramenta oficial do MySQL. É um ambiente completo que permite, além de realizar consultas, criar diagramas e trabalhar com engenharia reversa.

Com o *Workbench* devidamente instalado, chegou a hora de nos conectarmos à base de dados e criamos um novo *schema* para que possamos a partir dele criar as nossas tabelas de uma forma mais organizada. Para que seja possível a criação, deveremos efetuar o login na base de dados e após estarmos dentro do *Workbench*, devemos clicar no botão *create new schema*, conforme a Figura 1:

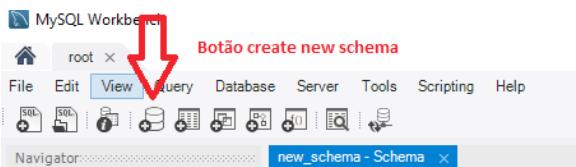


Figura 1 - Workbench

Fonte: os autores.

Após clicar neste botão será possível darmos um nome ao nosso novo *schema* de dados, conforme a Figura 2:

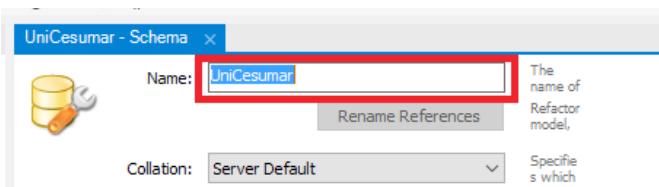


Figura 2 - Nome do *Schema*

Fonte: os autores.

Conforme vemos, o nome do nosso novo *schema* será **UniCesumar**, vale lembrar que essa notação será utilizada em todo o nosso desenvolvimento. Para que seja criado de fato esse novo *schema* será necessário clicar em *Apply* em que será mostrado uma tela com o comando SQL a ser aplicado no Banco de Dados para a criação.

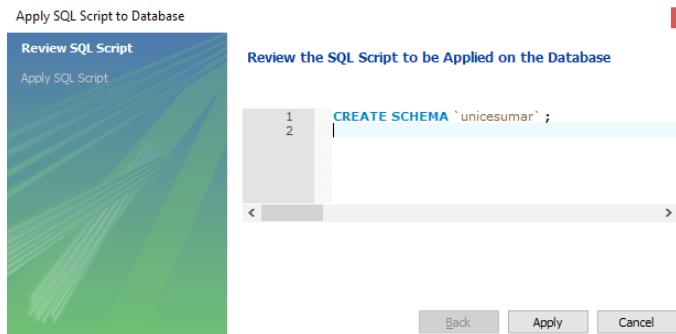


Figura 3 - Tela com o comando SQL  
Fonte: os autores.

Dessa maneira, basta clicar em *Apply* e depois em *Finish* para que o SGBD aplique o comando no Banco de Dados e crie o nosso novo *schema* que pode ser consultado por meio dos *schemas* presentes do lado esquerdo inferior no *Workbench*, conforme a Figura 4:

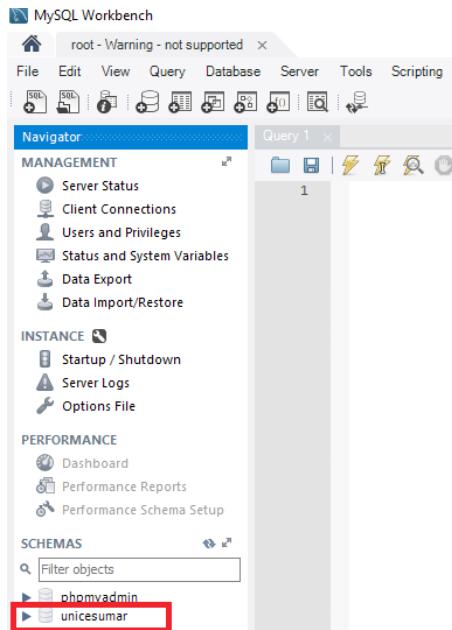


Figura 4 - *Workbench*  
Fonte: os autores.

A consulta aos *schemas* criados dentro do Banco de Dados pode acontecer de várias formas. Uma forma bem simples de verificar a criação do *schema* seria o

seguinte comando:

```
Show schemas
```

Esse comando deverá ser executado com script SQL no *Workbench*, para que seja aberto devemos clicar no botão  , logo abaixo de *File*. Após o editor de SQL aberto, basta digitar o comando e clicar no botão  para que o *Workbench* execute os comandos, após a execução. Os dados serão apresentados logo a seguir, conforme vemos na Figura 5:

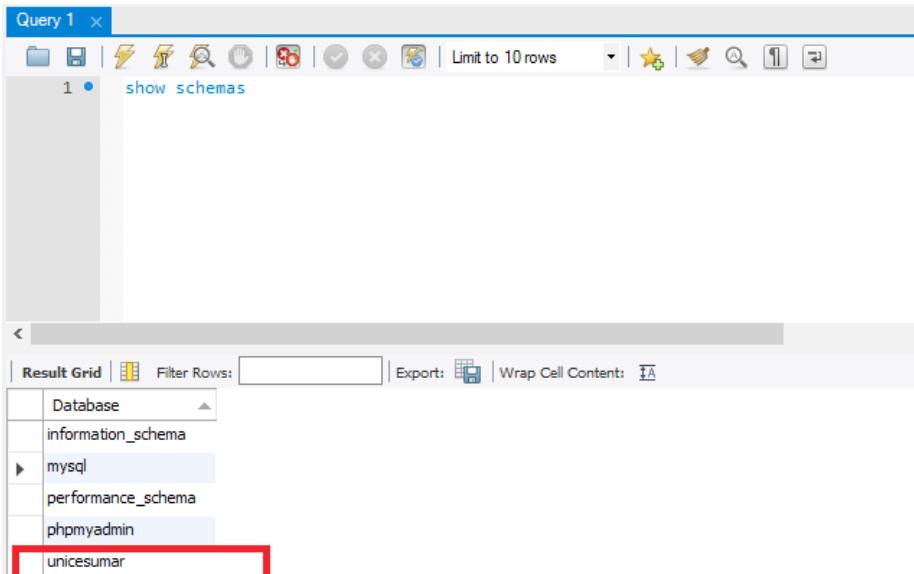


Figura 5 - Exemplo da execução.

Fonte: os autores.

Dessa forma, caro (a) aluno (a), você pode se perguntar: do que se trata a ferramenta *Workbench*? Ela é uma interface gráfica, que traduz comandos feitos pelo usuário em linguagem SQL, para que assim seja possível sua aplicação no Banco de Dados. Todas as ações feitas dentro do *workbench* podem ser substituídas de fato por comandos via terminal no *prompt* do dos, por exemplo.



## TIPOS DE DADOS

Para que possamos efetuar a criação de uma tabela, primeiro necessitamos saber mais um pouco sobre os tipos de dados. Em seguida, identificar os principais tipos de dados do MySQL, que será a ferramenta adotada para o desenvolvimento de todo o nosso conteúdo.

Conforme Passos (2010, on-line)<sup>1</sup>:

Alguns campos numéricos possuem a opção UNSIGNED. Isso quer dizer que o número não pode ser negativo. Por exemplo: o campo TINYINT com a opção UNSIGNED vai de 0 até 255, sem a opção UNSIGNED vai de -128 até 127.

A opção ZEROFILL completa o campo com zeros. Por exemplo: se você cadastrar '65' em um campo INT(5) – ou seja, inteiro com 5 dígitos – com a opção ZEROFILL habilitada, o MySQL irá cadastrar '000065'. Sem a opção ZEROFILL habilitada, o MySQL irá cadastrar '65'.

A	B	C	D	E
1	2	3	4	5
<b>Tipo</b>	<b>Classificação</b>	<b>Attributo</b>		
<b>TINYINT</b>	Número muito pequeno	Signed Unsigned	Tam. Min. -128,00 0,00	Tam. Max. 127,00 255,00
<b>SMALLINT</b>	Inteiro pequeno	Signed Unsigned	-32768,00 0,00	32767,00 65535,00
<b>MEDIUMINT</b>	Inteiro médio	Signed Unsigned	-8388608,00 0,00	8388607,00 16777215,00
<b>INT ou INTEGER</b>	Inteiro tamanho normal	Signed Unsigned	-2147483648,00 0,00	2147483647,00 4294967295,00
<b>BIGINT</b>	Inteiro grande	Signed Unsigned	-9223372036854770000,00 0,00	9223372036854770000,00 18446744073709500000,00
<b>L1</b>				
<b>L2</b>				
<b>L3</b>				
<b>L4</b>				
<b>L5</b>				
<b>L6</b>	<b>Classificação</b>	<b>Formato</b>	<b>Tam. Min.</b>	<b>Tam. Max.</b>
<b>DATE</b>	Data normal	'AAAA-MM-DD'	'1000-01-01'	'9999-12-31'
<b>DATETIME</b>	Data e hora	'AAAA-MM-DD HH:MM:SS'	'1000-01-01 00:00:00'	'9999-12-31 23:59:59'
<b>TIME</b>	Hora	'HH:MM:SS'	'-838:59:59'	'838:59:59'
<b>YEAR</b>	Ano	AA ou AAAA	'1000'	'9999'
<b>L7</b>				
<b>L8</b>				
<b>L9</b>				
<b>L10</b>				
<b>L11</b>				
<b>L12</b>				
<b>L13</b>				
<b>L14</b>				
<b>L15</b>				
<b>L16</b>	<b>Classificação</b>	<b>Classificação</b>	<b>Tam. Min.</b>	<b>Tam. Max.</b>
<b>VARCHAR</b>	String pequena com tamanho variável		1	255
<b>CHAR</b>	String pequena com tamanho fixo		1	255
<b>TEXT</b>	String		1	65535
<b>LONGTEXT</b>	String Grande		1	4.294.967.295
<b>L17</b>				
<b>L18</b>				
<b>L19</b>				
<b>L20</b>				
<b>L21</b>				
<b>L22</b>				
<b>L23</b>				
<b>L24</b>	<b>Classificação</b>	<b>Classificação</b>	<b>Tam. Min.</b>	<b>Tam. Max.</b>
<b>L25</b>	String pequena com tamanho variável		1	1
<b>L26</b>	String pequena com tamanho fixo		1	1
<b>L27</b>	String		1	65535
<b>L28</b>	String Grande		1	4.294.967.295

Tabela 1 - Tipos numéricos

Fonte: adaptado de Passos (2010, on-line)



SAIBA MAIS

## **Ranking de sistemas de bancos de dados mais usados em 2015/2016**

Sabemos dos inúmeros Bancos de Dados e seus SGBDs existentes no mercado, mas sempre vem aquela curiosidade em saber qual é o Banco de Dados mais utilizado. Para matar essa curiosidade a DB-Engines criou esse ranking.

DB-Engines é uma iniciativa de recolher e apresentar informações sobre os sistemas de gerenciamento de Banco de Dados.

O Ranking DB-Engines é uma lista de SGBDs classificados por sua atual popularidade. A lista é atualizada mensalmente. As propriedades mais importantes de numerosos sistemas são mostrados na visão geral de sistemas de gestão de Banco de Dados. Você pode examinar as propriedades de cada sistema, e pode compará-los lado a lado. Os termos e conceitos sobre Banco de Dados são explicados na enciclopédia acessível no site. DB-Engines foi criada e é mantida por solid IT.

Fonte: adaptado de DB-Engines (2016, on-line)2.

Reprodução proibida. Art. 184 do Código Penal e Lei 9610 de 19 de fevereiro de 1998

## CRIAÇÃO E ALTERAÇÃO DE TABELAS

O comando para a criação de tabelas é um comando Data Definition Language (DDL). Dessa forma, iremos trabalhar com as tabelas de dados formando um cadastro de pessoa, assim podemos representar graficamente, conforme a Figura 6:



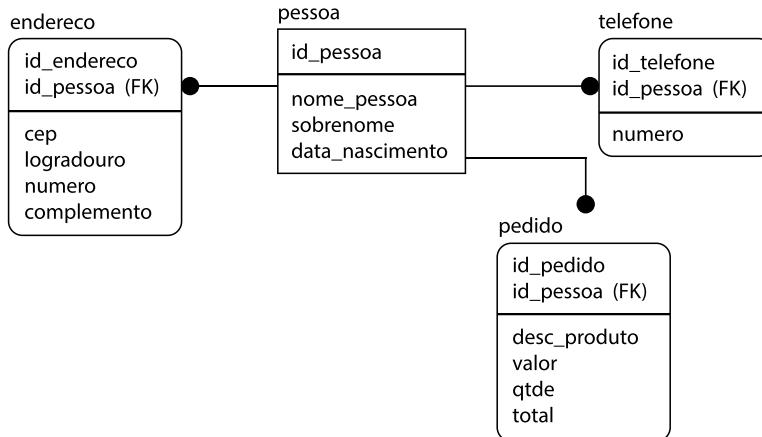


Figura 6 - Comandos para criação de tabelas

Fonte: os autores.

De acordo com a figura, vamos ter em nossa base de dados um “cadastro” de pessoas que poderão ter vários endereços e vários telefones.

No caso de você aluno(a), sugiro que siga todos os passos deste livro utilizando o MySQL, pois no final estaremos todos com os mesmos dados no Banco de Dados. Nossa próximo passo agora é a criação e a alteração das tabelas em nosso Banco de Dados.

A criação de tabelas se dá a um comando DDL. Para evidenciar melhor na prática como é de fato a criação e a alteração de tabelas em um Banco de Dados, os comandos a seguir podem ser executados de fato em seu Banco de Dados, a fim de que nossa disciplina fique mais prática. A sequência de comandos a seguir, irá criar em nossa base de dados uma tabela chamada pessoa a qual estará ligada com outras duas tabelas, endereços e telefones. Seguindo o raciocínio, nossa intenção é ter uma pessoa cadastrada em nossa base que possa conter vários endereços e vários telefones:



### Criação da tabela pessoa:

```
CREATE TABLE 'unicesumar'.'pessoa' ('id_pessoa' INT NOT NULL,  
          'nome' VARCHAR(200) NOT NULL,  
          'sobrenome' VARCHAR(200) NOT NULL,  
          'data_nascimento' DATE NULL  
        ) ;
```

### Criação da tabela endereço:

```
CREATE TABLE 'unicesumar'.'endereco' ('id_endereco' INT  
NOT NULL,  
          'cep' INT(8) NOT NULL,  
          'logradouro' VARCHAR(200) NULL,  
          'numero' VARCHAR(20) NULL,  
          'compimento' VARCHAR(200) NULL  
        ) ;
```

### Criação da tabela telefone:

```
CREATE TABLE 'unicesumar'.'telefone' ('id_telefone' INT NOT  
NULL,  
          'numero' BIGINT(12) NOT NULL  
        ) ;
```

## CHAVE PRIMÁRIA

As criações de nossas tabelas foram feitas sem a identificação de uma chave primária, a função da chave primária em uma tabela no Banco de Dados é a de que nunca haverá a repetição de um mesmo valor para esse campo.

Visto tal conceito, iremos alterar as nossas tabelas modificando os campos que iniciam com “id\_” para que sejam nossas chaves primárias.



### Alteração da tabela pessoa:

```
ALTER TABLE 'unicesumar'.'pessoa'  
ADD PRIMARY KEY ('id_pessoa');
```

### Alteração da tabela endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD PRIMARY KEY ('id_endereco');
```

### Alteração da tabela telefone:

```
ALTER TABLE 'unicesumar'.'telefone'  
ADD PRIMARY KEY ('id_telefone');
```



Agora, com os comandos executados em seu Banco de Dados, temos as três tabelas devidamente criadas, mas sem dependência entre elas. Ou seja, todas estão distintas no Banco de Dados. Para solucionar esses problemas e trabalharmos melhor com a integridade dos dados, iremos criar mais uma coluna (`id_pessoa`) nas tabelas de endereço e telefone. As tabelas mencionadas irão fazer parte da criação de nossa *foreign key* sendo referenciadas pelo campo identificador da tabela de pessoa:

#### Alteração da tabela de endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD COLUMN 'id_pessoa' INT NOT NULL;
```

#### Alteração da tabela de telefone:

```
ALTER TABLE 'unicesumar'.'telefone'  
ADD COLUMN 'id_pessoa' INT NOT NULL;
```

Agora com as tabelas prontas, podemos criar referências entre as tabelas endereço e telefone com a tabela de pessoa. Para isso, devemos efetuar a criação da *foreign key* na tabela de endereço e telefone referenciando a tabela de pessoa:

#### Criação da foreign key na tabela de endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD CONSTRAINT 'pessoa_fk_endereco'  
FOREIGN KEY ('id_pessoa')  
REFERENCES 'unicesumar'.'pessoa' ('id_pessoa');
```

## Criação de foreign key na tabela de telefone:

```
ALTER TABLE 'unicesumar'.'telefone'
ADD CONSTRAINT 'pessoa_fk_telefone'
FOREIGN KEY ('id_pessoa')
REFERENCES 'unicesumar'.'pessoa' ('id_pessoa');
```

Dessa maneira, estamos garantindo a integridade de nosso Banco de Dados, a fim de que todas as ligações tenham efeito. Antes de todas as alterações nas tabelas tínhamos as tabelas distribuídas em nosso Banco de Dados conforme a Figura 7:

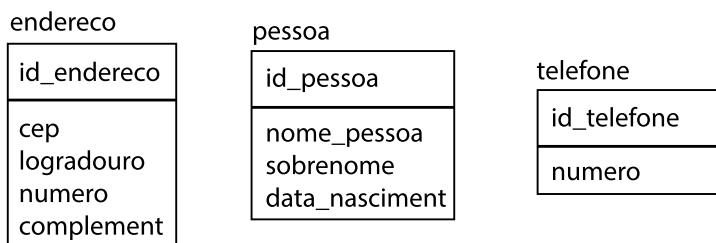


Figura 7 - Exemplo

Fonte: os autores

Após todas as alterações aplicadas em nossa base de dados, criando assim a integridade em nossos dados, eles que antes não tinham nenhum tipo de ligação, agora formam o Modelo Entidade Relacionamento (MER) da Figura 8:



Figura 8 - MER

Fonte: os autores.

## POPULANDO AS TABELAS CRIADAS

O comando `INSERT` faz parte dos comandos de *Data Manipulation Language* (DML). Nesta etapa, iremos popular nossas tabelas de acordo com a criação e respeitando a integridade dos dados. Sempre que gravamos dados em tabelas devemos obedecer a ordem hierárquica delas. Veremos a seguir o que seria essa ordem.

Para inserir os dados nas tabelas, vamos iniciar pela tabela de pessoa, em que iremos inserir um registro somente:

```
INSERT INTO 'unicesumar'.'pessoa'  
        ('id_pessoa',  
         'nome',  
         'sobrenome',  
         'data_nascimento')  
  
VALUES  
        (1, /*campo id_pessoa*/  
         'João', /*campo nome */  
         'Pereira', /*campo sobrenome */  
         '1983-09-15'); /* campo data_nascimento
```



Com os dados inseridos em nossa base, na tabela de pessoa, agora iremos adicionar dois endereços referenciando essa pessoa cadastrada. Com a integridade entre as tabelas criadas adequadamente, podemos ter vários endereços para uma única pessoa e um endereço poderá estar ligado a apenas uma pessoa. Temos assim uma cardinalidade de N:1 partindo da tabela de pessoa. Atenção no código a seguir temos comentários sobre os campos do código que estão entre `/* */`

## Inserindo o endereço 1:

```
INSERT INTO 'unicesumar'.'endereco'  
    ('id_endereco',  
     'cep',  
     'logradouro',  
     'numero',  
     'complemento',  
     'id_pessoa')  
  
VALUES  
(1, /*campo id_endereco*/  
 86890880, /*campo cep*/  
 'Av. Paraná', /*campo logradouro*/  
 '27356', /*campo numero*/  
 'Casa', /*campo complemento*/  
 1); /*campo id_pessoa FK com a tabela de pessoa*/
```

## Inserindo o endereço 2:

```
INSERT INTO 'unicesumar'.'endereco'  
    ('id_endereco',  
     'cep',  
     'logradouro',  
     'numero',  
     'complemento',  
     'id_pessoa')  
  
VALUES  
(2, /*campo id_endereco*/  
 86890880, /*campo cep*/  
 'Av. Brasil', /*campo logradouro*/  
 '412 sala 2', /*campo numero*/  
 'Comercial', /*campo complemento*/  
 1); /*campo id_pessoa FK com a tabela de pessoa*/
```

Agora, nós já temos uma pessoa em nossa base que possui dois endereços. Por final, iremos inserir os telefones pertencentes a essa pessoa:

### Inserindo o telefone 1:

```
INSERT INTO 'unicesumar'.'telefone'  
    ('id_telefone',  
     'numero',  
     'id_pessoa')  
VALUES  
    (1, /* campo id_telefone */  
     4499786543, /* campo numero*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

### Inserindo o telefone 2:

```
INSERT INTO 'unicesumar'.'telefone'  
    ('id_telefone',  
     'numero',  
     'id_pessoa')  
VALUES  
    (2, /* campo id_telefone */  
     4499786523, /* campo numero*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

A partir dos dados inseridos nas tabelas, podemos tirar algumas conclusões. A pessoa por nome de João possui dois endereços, um na Av. Paraná e outra na Av. Brasil, sendo o primeiro o endereço de sua casa, e outro de sua empresa. Além disso, ele possui 2 números de telefone.

Um forte elemento para a aprendizagem da linguagem SQL é a prática. Desse modo, sugiro a você que efetue mais inserções no Banco de Dados seguindo os comandos exemplificados anteriormente, sempre lembrado de que não é possível ter um valor de uma chave primária repetida na tabela.



## COMANDO DESCRIBE

É de extrema importância consultar a estrutura das tabelas já criadas em um Banco de Dados. Essa consulta é efetuada por meio do comando **describe**. Com ele é possível visualizar as colunas e os tipos de colunas de uma tabela em específico.

Para o uso deste comando é necessário efetuar a interpretação dos dados retornados, pois será a partir desses dados que você saberá tudo sobre a tabela. Destes dados retornados podemos evidenciar: colunas de índices, NOT NULL, NULL, PRIMARY KEY, quais os campos compõem a PRIMARY KEY (chave primária). Também qual é o valor *default* (padrão) para determinada coluna, se a PRIMARY KEY é auto incrementada pelo próprio Banco de Dados. Vale lembrar que o comando DESC é um sinônimo do comando DESCRIBE.

Para utilizar esse comando devemos obedecer a sintaxe a seguir:

```
DESCRIBE nome_da_tabela;
```

Vamos exemplificar esse comando com a tabela pessoa, a qual já criamos nesta mesma unidade. O comando a ser executado deverá ser:

```
describe pessoa;
```

O retorno dos dados para esse comando deverá ser uma linha para cada coluna da tabela conforme Figura 9:

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
id_pessoa	int(11)	NO	PRI	NULL	
nome	varchar(200)	NO		NULL	
sobrenome	varchar(200)	NO		NULL	
data_nascimento	date	YES		NULL	

Figura 9 - Modelo de uma linha para cada coluna

Fonte: os autores.

As informações mais importantes sobre o retorno desse comando são:

**Field:** essa informação representa o nome das colunas presentes na tabela do Banco de Dados.

**Type:** indica o tipo de dado de cada coluna da tabela.

**Null:** o conceito é bem simples, indica a obrigatoriedade da coluna, ou seja, se ela aceita valores nulos, representados pelo retorno YES e NO. Como no exemplo anterior, apenas o campo data\_nascimento poderá ser nulo, os demais são campos obrigatórios.

**KEY:** esse indicador poderá retornar apenas 3 valores: PRI, UNI, MUL

- **PRI:** indica que a coluna é uma chave primária da tabela ou faz parte da composição juntamente com outros campos, caso seja uma chave primária composta o valor PRI estará presente em mais campos da tabela, referenciadas na coluna KEY.
- **UNI:** é um campo UNIQUE + NOT NULL, esse tipo de campo é muito parecido com a chave primária da tabela, pois toda chave primária da tabela não pode ser nula e deve ser única. Esse conceito de UNI é utilizado para que não existam repetições de valores nesse campo presente na tabela.
- **MUL:** é a definição de uma coluna que é ou é parte de um índice não único que aceitam valores múltiplos e NULL.

**DEFAULT:** é utilizado para que seja possível o Banco de Dados inserir um valor padrão caso esse valor não seja informado no momento do *insert*.

Apesar de um comando muito simples o DESCRIBE é um dos recursos disponíveis mais importantes do MySQL. Por isso, conhecê-lo e saber interpretar os seus resultados é de extrema importância. No dia a dia dos programadores de Banco de Dados esse é um dos comandos mais utilizados.



REFLITA

Um Banco de Dados que é elaborado sem levar em consideração as regras de negócio da empresa, poderá passar por problemas futuramente?

## CONSIDERAÇÕES FINAIS

Prezado(a) aluno(a)! Nesta unidade procuramos demonstrar na prática a utilidade dos conceitos até então abordados. Trabalhamos na prática os comando DDL e DML. Espero que essa prática tenha evidenciado a você como é prazeroso trabalhar com Banco de Dados, compreendendo os princípios teóricos que abrangem este grande mercado de trabalho. Dessa maneira, procuramos exemplos da prática. Sugiro que caso não tenha seguidos os passos, execute como exercício cada comando destacado em itálico nos quadros desta unidade, pois o passo a passo criará sua base de dados, as tabelas e irá populá-las.

Todos os conceitos abordados fazem parte do dia a dia de um programador e é essencial para o desenvolvimento de qualquer programa que trabalhe em conjunto com um Banco de Dados.

Nos vemos na próxima unidade, até lá!

## ATIVIDADES



1. O comando alter table se encontra em qual definição?
  - a) DML
  - b) DTL
  - c) DQL
  - d) DDL
  - e) DCL
2. Quais as vantagens da criação de um **schema**?
3. O comando insert se encontra em qual definição?
  - a) DML
  - b) DTL
  - c) DQL
  - d) DDL
  - e) DCL
4. Sobre os tipos de dados assinale, verdadeiro ou falso para os itens relacionados abaixo:  
(      ) **DATETIME**: combinação de data e hora no formato  
(      ) **YEAR**: ano com 4 dígitos  
(      ) **LONGBLOB**: string com até 4Gb  
(      ) **DECIMAL**: não armazena números inteiros
  - a) V, V, V, F
  - b) V, F, V, F
  - c) V, V, F, F
  - d) F, V, F, V
  - e) F, F, V, V
5. Popule mais dados da tabela pessoa seguindo o item 5 desta unidade.

# MATERIAL COMPLEMENTAR



LIVRO

## **Projeto de Banco de Dados - Vol.4 (2008)**

Carlos Alberto Heuser

**Editora:** Bookman

**Sinopse:** em sua sexta edição e adotado por faculdades de todo o Brasil, Projeto de Banco de Dados aborda as duas primeiras etapas do ciclo de vida de um Banco de Dados: modelagem conceitual e projeto lógico.



# REFERÊNCIAS

## REFERÊNCIAS ON-LINE

- 1- Em: <<http://blog.tiagopassos.com/author/admin/page/37>>. Acesso em: 18. nov. 2016.
- 2- Em: <<http://db-engines.com/en/>>. Acesso em: 18. nov. 2016.



# GABARITO

1. A

2. Agrupamento de vários objetos de um ou mais usuários de Banco de Dados como exemplo: tabelas, sequências, índices etc. São associados a um Banco de Dados na razão de vários esquemas para um Banco de Dados, facilitando a administração dos objetos e dos dados. Principais vantagens da utilização de *schemas* em Banco de Dados são as permissões que se pode atribuir e revogar ao usuário. Pois podemos, dessa maneira, ter vários *schemas* e vários usuários em um Banco de Dados, mas determinados usuários terão acesso a somente um *schema* no Banco de Dados.

3. A

4. A - V, V, V, F

**5. *INSERT INTO* 'unicesumar'.'pessoa'**

```
('id_pessoa',
  'nome',
  'sobrenome',
  'data_nascimento')
```

**6. *VALUES***

```
(select max(id_pessoa) + 1 from pessoa,
      'Maria', /*campo nome */
      'Pereira', /*campo sobrenome */
      '1987-09-15'); /* campo data_nascimento
```





# A LINGUAGEM SQL

## Objetivos de Aprendizagem

- Compreender cada divisão da linguagem SQL.
- Importância e ligações entre as divisões da linguagem SQL.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- História da SQL
- Linguagem de Consulta de Dados (DQL)
- Linguagem de Manipulação de Dados (DML)
- Linguagem de Definição de Dados (DDL)
- Linguagem de Controle de Dados (DCL)
- Linguagem de Transação de Dados (DTL)



## INTRODUÇÃO

Olá caro(a) aluno(a)! Nesta unidade serão descritas as linguagens formais que irão proporcionar uma notação concreta para o desenvolvimento de consultas, alterações e remoções em um Banco de Dados.

Contudo, os Banco de Dados comerciais necessariamente precisam de uma linguagem fácil para o usuário. Nesta unidade do livro iremos trazer para você os principais conceitos da linguagem SQL. A sigla SQL que vem do inglês *Structured Query Language* - Linguagem de Consulta Estruturada.

Embora estamos falando da linguagem SQL como uma linguagem de consulta de dados, ela tem várias outras funções. Ela também é utilizada para a definição da estrutura de dados, modificações do Banco de Dados, especificações de segurança, dentre outras.

Neste livro, não temos a intenção de passar detalhadamente tudo sobre SQL para você, vamos apenas trabalhar os construtores e os principais conceitos da linguagem SQL. Pois essa linguagem certamente é a linguagem padrão dos Bancos de Dados relacionais, devido a sua simplicidade e facilidade de uso. A SQL é a linguagem de mais alto nível para a manipulação de dados dentro do modelo relacional. (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Bom estudo!



## HISTÓRIA DA SQL

Tudo começou no início dos anos 70, por meio de uma pesquisa da empresa *International Business Machine* (IBM) com o Dr. E. F. Codd o qual levou ao desenvolvimento de um “produto” chamado SEQUEL ou Linguagem de Consulta em Inglês Estruturado, posteriormente a SEQUEL veio a se transformar na SQL ou Linguagem de Consulta Estruturada.

A IBM e os demais fornecedores de Banco de Dados relacionais querem padronizar a forma de acessar e manipular seus dados em um Banco de Dados relacional. Embora a IBM tenha sido a primeira a desenvolver esse tipo de tecnologia, foi a Oracle quem lançou comercialmente o Banco de Dados relacional.

## SQL COMO UMA LINGUAGEM PADRÃO

Um grande padrão de Banco de Dados é a linguagem SQL decorrente de sua simplicidade e facilidade de uso. Diferencia-se das demais linguagens de consulta de Banco de Dados no sentido em que se é especificado a forma do resultado, sem ter nenhum tipo de preocupação com o percurso percorrido para se chegar ao resultado. A SQL tem o seu ciclo de aprendizado menor que as demais linguagens de programação, pois ela é uma linguagem declarativa o que se opõem às demais linguagens de programação procedurais.

Embora a SQL tenha sido criada pela IBM, ocorreram várias derivações dessa linguagem criadas por outros autores. Visto o crescimento da linguagem

foi necessário a criação de uma padronização para a linguagem. Essa tarefa ficou a cargo do *American National Standards Institute (ANSI)* no ano de 1986 e pela *International Organization for Standards (ISO)* em 1987. A SQL foi revista em 1992 e essa versão recebeu o nome de SQL-92, revisto novamente em 1999 levando o nome de SQL-99 ou também chamado de SQL-3. Embora existam essas padronizações ANSI e ISO, ela ainda possui variações e extensões produzidas pelos diferentes fabricantes de Sistema de Gerenciamento de Banco de Dados (SGBD).

A linguagem SQL pode ser definida em várias partes:

- DML - *Data Manipulation Language* (Linguagem de Manipulação de Dados). Esse comando é utilizado para realizar inclusões, exclusões e alterações de dados, os quais são utilizados a partir dos comandos INSERT, UPDATE e DELETE. (WATSON; RAMKLAAS, 2012)
- DDL - *Data Definition Language* (Linguagem de Definição de Dados). Conjunto de comandos dentro da SQL que permitem ao desenvolvedor utilizá-las para a definição das estruturas de dados, contendo instruções que permitem a criação, modificação e remoção de tabelas, bem como criação, alteração e remoção de elementos associados às tabelas. (SILBERSCHATZ; KORTH; SUDARSHAN, 1999)
- DCL - *Data Control Language* (Linguagem de Controle de Dados). São os comandos que permitem ao administrador do Banco de Dados gerenciar as autorizações dos dados e licenças dos usuários para controlar o acesso de quem pode ver ou manipular dados dentro do banco de dados. Os comandos mais utilizados são: GRANT, REVOKE, SET e LOCK.
- DTL - *Data Transaction Language* (Linguagem de Transação de Dados). São os comandos utilizados para gerenciar as mudanças feitas nos dados do Banco de Dados através de um comando DML, ele permite que as instruções sejam agrupadas por transações. Os principais comandos são: ROLLBACK, COMMIT e SAVEPOINT.
- DQL - *Doctrine Query Language* (Linguagem de Consulta de Dados). Embora com apenas um comando, se não o mais importante de todos os comandos utilizados na SQL. O SELECT permite ao usuário efetuar uma consulta no Banco de Dados. Esse comando é composto de várias cláusulas e opções, possibilitando das consultas mais simples às consultas mais complexas. (DOCPLAYER, on-line, [2016])<sup>1</sup>



## DQL - LINGUAGEM DE CONSULTA DE DADOS

A estrutura de um *select* simples consiste em três cláusulas: *select*, *from* e *where*.

- **Cláusula SELECT:** é responsável por relacionar os atributos desejados no resultado de uma consulta, ou seja, ele permite recuperar os dados de uma tabela do Banco de Dados. Esse comando corresponde à operação da álgebra relacional.
- **Cláusula FROM:** é obrigatória em toda instrução select, pois é ela a responsável por associar as relações que serão pesquisadas durante a evolução da expressão, é ela a responsável por ligar o campo que iremos recuperar no select com a tabela no Banco de Dados. Essa cláusula corresponde à operação de produto cartesiano da álgebra relacional.
- **Cláusula WHERE:** é responsável por estabelecer uma condição de pesquisa, ou seja, tem a função de filtrar os dados a serem recuperados do Banco de Dados. A cláusula where também é utilizada nos comandos de UPDATE, DELETE. Ela corresponde à seleção do predicado da álgebra relacional.

## CLÁUSULA SELECT

A declaração SELECT é um mecanismo extremamente elegante, fácil de ser trabalhado e altamente extensível. O mecanismo foi criado com o intuito de recuperar os dados existentes no Banco de Dados, afinal de contas não teria lógica ter dados gravados em um Banco de Dados se não fosse possível fazer a leitura deles. As cláusulas do SELECT vem do inglês, e se pensarmos a partir da tradução ficará mais simples ainda a compreensão. As cláusulas são:

SELECT: em português, “selecionar”

FROM: em português, “a partir de”

WHERE: em português, “onde”

Os dados retornados por uma consulta SQL são, naturalmente, uma relação, pensando em uma consulta simples. Segue o exemplo de um SELECT simples:

```
SELECT nome_pessoa  
      FROM     pessoa
```

A partir desta consulta, iremos encontrar o nome de todas as pessoas da tabela pessoa do Banco de Dados. Esse resultado é uma relação de um campo simples titulado de nome\_pessoa na tabela pessoa. A linguagem SQL em sua maioria permite duplicidade em suas relações, podendo trazer assim dados duplicados no resultado de sua consulta, no caso para forçarmos a eliminação dos resultados duplicados utilizamos a instrução **distinct** depois da cláusula *select*, e, que se reescrevermos a consulta anterior, ficará da seguinte maneira:

```
SELECT DISTINCT nome_pessoa  
      FROM     pessoa
```

No mesmo formato anterior também é possível a utilização do asterisco (\*). Ele é usado para denotar que o *select* deverá trazer todos os campos disponíveis para consulta. A consulta com o uso do asterisco ficará da seguinte maneira:

```
SELECT DISTINCT *
FROM pessoa
```

Dessa maneira, estamos solicitando ao Banco de Dados que seja apresentado todos os campos disponíveis na tabela *pessoa* e que não possua dados repetidos.

## WHERE

O WHERE é utilizado para delimitar os dados a serem retornados pela nossa consulta, os filtros a serem aplicados se restringem às linhas utilizando operadores de comparação em um conjunto de campos e valores literários. Os filtros trabalham com os operadores booleanos que fornecem um mecanismo para especificar condições múltiplas para filtrar as linhas a serem retornadas. Segue o exemplo de como ficaria a implementação do *where* na consulta simples:

```
SELECT *
FROM pessoa
WHERE codigo_pessoa = 1
```

```
SELECT *
FROM pessoa
WHERE nome_pessoa = 'JOAO'
```

O primeiro exemplo trata de uma consulta para trazer os dados da tabela pessoa, desde que exista uma pessoa com o código da pessoa que contenha o número um. No segundo exemplo o resultado depende extremamente do nome da pessoa ser JOAO.

### Condições baseadas em Números

As condições devem ser propriamente utilizadas de acordo com o tipo de dados a serem filtrados. Vale lembrar que nas condições numéricas não é necessário colocar a restrição entre aspas simples. Segue um quadro com os possíveis operadores para as condições numéricas:

Quadro 1 - Possíveis operadores para as condições numéricas

OPERADOR	DESCRIÇÃO
<	Menor que
>	Maior que
<=	Menor que ou igual a
>=	Maior que ou igual a
<>	Não igual
!=	Não igual
=	Igual a

Fonte: os autores

A partir desse quadro podemos elaborar os filtros que se aplicam a nossa necessidade.

### Condições baseadas em Caracteres

As condições determinantes para as linhas selecionadas, baseadas em caracteres se dão pelo fato de delimitar os caracteres dentro de aspas únicas, em que a não utilização das aspas gera um erro em sua execução. Lembre que existe a distinção entre maiúsculos e minúsculos. Exemplo: *where nome = 'joão'* é diferente de *where nome = 'João'*.

## Condições baseadas em Data

Os campos de tipo data são muito úteis para o armazenamento de datas e horas. As datas quando utilizadas nos filtros devem seguir o mesmo padrão utilizados para os caracteres sendo delimitadas por aspas única.

## FROM

Finalmente, vamos falar da cláusula FROM, ela é responsável por fazer a ligação com a tabela que vamos efetuar a consulta. Em resumo, ela especifica as tabelas que possuem as colunas listadas na cláusula SELECT. Por exemplo, para trazermos em uma consulta todos os dados da tabela pessoa, teríamos que especificar a tabela pessoa na cláusula FROM do SELECT conforme exemplo a seguir:

```
SELECT *
FROM     pessoa
```

Nesse exemplo, estamos trazendo todos os dados da tabela pessoa.



## DML - LINGUAGEM DE MANIPULAÇÃO DE DADOS

Caro(a) aluno(a), como vimos anteriormente, DML é a Linguagem de manipulação de dados e são os comandos dentro da linguagem SQL utilizados para a inclusão, remoção e alteração de dados em um Banco de Dados. Como já vimos os principais comandos são: INSERT, UPDATE e DELETE.

Segundo John Watson e Roopesh Ramklass (2009) a maioria dos profissionais não incluem o *select* como sendo um comando DML, pois ele é considerado uma linguagem separada em seu próprio direito.

### INSERT

O comando INSERT é simples, digamos que ele é um pedido de inclusão de uma linha em uma tabela. Vale lembrar que os valores a serem inseridos no Banco de Dados devem pertencer ao domínio do campo. Contudo as ordens dos campos devem seguir a ordem de criação dos campos na tabela. As linhas podem ser preenchidas de diversas maneiras, mas a maneira mais utilizada e mais simples é o INSERT. As versões mais básicas da instrução inserem apenas uma linha em uma tabela, mas as variações mais complexas podem chegar a inserir várias linhas em várias tabelas.

Por meio do comando INSERT podemos inserir uma ou mais linhas no Banco de Dados. Segue um exemplo da instrução INSERT em sua variação mais simples.

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ....  
colunaN)  
VALUES (valor1, valor2, valor3, ..., valorN);
```

## UPDATE

O comando UPDATE é utilizado para efetuar alterações nas linhas já existentes no Banco de Dados que possivelmente foram gravadas por meio do comando INSERT. Da mesma maneira que o INSERT, o UPDATE pode afetar uma linha somente ou um conjunto de linhas existentes na tabela. Para que se possa delimitar a linha ou quais linhas receberão a alteração é utilizado a cláusula WHERE que recebe os mesmos tratamentos do SELECT.

Um grande diferencial do UPDATE é que não é possível atualizar campos de mais de uma tabela de uma única vez. Ao atualizar uma linha ou várias linhas o UPDATE determina quais colunas atualizar, e assim, não é necessário atualizar cada coluna da linha. Poderá ser alterado apenas uma coluna da linha. Caso a coluna já estiver preenchida com algum valor, o valor antigo será substituído pelo novo valor passado pelo comando UPDATE. Caso a coluna estiver vazia ela será preenchida depois do UPDATE com o novo valor. A sintaxe básica é a seguinte:

```
UPDATE nome_da_tabela SET coluna = 'novo_valor'
```

## DELETE

As linhas inseridas e alteradas com os exemplos anteriores agora poderão ser removidas por meio do comando DELETE. Esse comando pode remover uma linha ou um conjunto de linhas da tabela, a quantidade de linhas a serem removidas irá depender da cláusula WHERE. Caso a cláusula não for utilizada, o comando irá remover todas as linhas pertencentes à tabela, o que pode ser um

problema se você esquecer de colocar a cláusula accidentalmente. Uma exclusão é tudo ou nada, pois não se pode definir colunas para a remoção e sim a linha inteira. A sintaxe mais simples para a remoção de uma linha, é a seguinte:

```
DELETE nome_da_tabela;
```

Por meio desse comando iremos remover todos os dados da tabela informada. O efeito de uma instrução DML não é permanente até que você confirmar a transação, que o incluir. A **transação** é uma sequência de instruções SQL, podendo ser uma única instrução DML.

Até que uma transação seja confirmada, ela pode ser revertida (desfeita). Para mais informações sobre transações, consulte “Sobre instruções de controle de transação”.

## DDL - LINGUAGEM DE DEFINIÇÃO DE DADOS

Como já tratamos no início desta unidade, DDL ou em inglês *Data Definition Language* é um conjunto dos comandos da linguagem SQL utilizado para a definição das estruturas dos dados, fornecendo as instruções para criação, modificação e remoção das tabelas do Banco de Dados.



## VALORES NOT NULL

Na linguagem SQL podemos definir que uma tabela poderá conter campos com valores nulos. O único campo que não poderá ficar nulo é a chave primária, caso seja obrigatório o preenchimento, deve-se criar esse campo com o atributo NOT NULL, pois assim reforçamos a necessidade de se informar um valor para um determinado campo na tabela. Dessa maneira, o campo especificado não poderá ser deixado de informar, pois não pode ser deixado em branco, ou seja, não poderá conter NULL em seu valor.

## CHAVES E INTEGRIDADE

### Chaves primárias

Para que seja possível especificar uma chave primária, utiliza-se a cláusula PRIMARY KEYS (em inglês, *Primary keys* ou “PK”). Uma chave primária tem a função de tornar-se um registro em uma tabela única, sendo assim nunca haverá a repetição de um mesmo valor para este campo. A chave primária pode ser atribuída a um ou mais campos, considerando dessa maneira que nunca haverá a repetição da combinação desses dois valores na mesma tabela no Banco de Dados. Segue a sintaxe para a utilização da PRIMARY KEYS, lembrando que esses comandos vão ao final da sintaxe de criação das tabelas:

```
CREATE TABLE nome_da_tabela (
    nome_do_campo INT NOT NULL
    PRIMARY KEY (nome_do_campo);
```

A chave estrangeira acontece quando um campo de uma tabela for chave primária em outra tabela. Podemos pensar da seguinte maneira, sempre que houver o relacionamento 1:N entre duas tabelas, a tabela 1 receberá a chave primária e a tabela N receberá a chave estrangeira. A seguir, a sintaxe para a criação:

```
CREATE TABLE nome_da_tabela(
    'id_tabela' INT NOT NULL,
    'campo1' VARCHAR(45) NOT NULL,
    'campo2' INT NOT NULL,
    PRIMARY KEY ('id_tabela'),
    CONSTRAINT 'campo2'
        FOREIGN KEY ('campo2')
        REFERENCES 'tabela_referenciada' ('campo_referenciado')
        ON DELETE NO ACTION
        ONUPDATE NO ACTION);
```

## CRIANDO UMA TABELA SIMPLES

Existem várias maneiras de se armazenar uma tabela no Banco de Dados, porém a mais simples e mais utilizada é a tabela empilhada. Essa pilha são as linhas. O seu comprimento pode ser variado de forma aleatória, podendo haver uma correlação entre as linhas inseridas com a ordenação de armazenamento. Para a criação das tabelas é utilizado o comando CREATE TABLE, em que o primeiro parâmetro deste comando é o nome da tabela, seguido dos campos, seus respectivos tipos e suas devidas restrições. Por exemplo:

```
CREATE TABLE schema.nome_da_tabela (nome_da_coluna tipo_da_
coluna);
```

Obrigatoriamente, se deve especificar o nome da tabela a ser criada. Essa tabela deverá conter no mínimo um campo, para que seja possível sua criação. Segue, o exemplo para a criação da tabela pessoa e telefone da Figura 1:

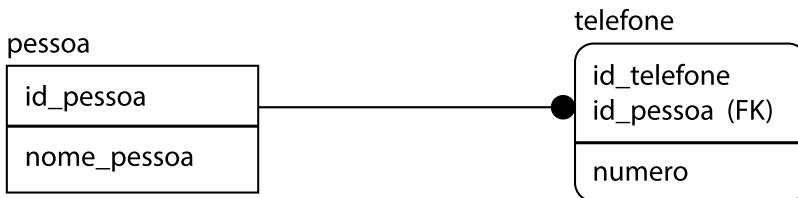


Figura 1 - Exemplo para a criação da tabela

Fonte: os autores.

Os comandos a seguir são responsáveis por criar as tabelas definidas na figura:

```

CREATE TABLE 'pessoa' (
    'id_pessoa' INT NOT NULL,
    'nome_pessoa' VARCHAR(100) NOT NULL,
    'peso' DECIMAL(10,2) NULL,
    PRIMARY KEY ('id_pessoa'));
    
```

```

CREATE TABLE 't telefone' (
    'id_telefone' INT NOT NULL,
    'numero' VARCHAR(45) NOT NULL,
    'id_pessoa' INT NOT NULL,
    PRIMARY KEY ('id_telefone'),
    CONSTRAINT 'id_pessoa'
        FOREIGN KEY ('id_pessoa')
        REFERENCES 'pessoa' ('id_pessoa')
        ON DELETE NO ACTION
        ON UPDATE NO ACTION);
    
```

## ALTERANDO AS DEFINIÇÕES DE UMA TABELA JÁ EXISTENTE

Existem muitas alterações que podem ser efetuadas em uma tabela após a sua criação, existem muitas alterações em seu meio físico. Essas são de responsabilidade do administrador de Banco de Dados, mas muitas outras são puramente lógicas e poderão ser feitas por meio dos desenvolvedores SQL, seguem exemplos das alterações possíveis:

### Adicionar uma coluna

```
alter table nome_da_tabela add (nome_do_campo tipo_do_campo);
```

### Modificando uma coluna

```
alter table nome_da_tabela modify (nome_do_campo novo_tipo_campo);
```

### Deletando uma coluna

```
alter table nome_da_tabela drop column nome_do_campo;
```

### Adicionando uma restrição a coluna

```
ALTER TABLE nome_da_tabela ADD FOREIGN KEY (tabela1_fk_tabela2) REFERENCES tabela_referenciada;
```

### Para adicionar a restrição de não nulo

```
ALTER TABLE nome_da_tabela ALTER COLUMN nome_do_campo SET NOT NULL;
```

### Removendo restrição

```
ALTER TABLE produtos DROP CONSTRAINT nome_da_restricão;
```

### Removendo restrição *not null*

```
ALTER TABLE produtos ALTER COLUMN cod_prod DROP NOT NULL;
```

## DELETANDO UMA TABELA

O comando `DROP TABLE` permite a remoção de uma tabela do Banco de Dados. Essa operação remove linhas, estrutura e índices de acesso associados à tabela. Esse comando é bem simples em relação aos demais, segue a sintaxe:

```
drop table nome_da_tabela;
```

## DCL - LINGUAGEM DE TRANSAÇÃO DE DADOS

A DCL, conhecida também como Linguagem de Controle de Dados, permite o controle de acesso e manipulação a dados dentro do Banco de Dados. Normalmente esses comandos são utilizados para controlar a distribuição de privilégios entre um usuário e outro.



SAIBA MAIS



### O que é computação nas nuvens (cloud computing)?

O termo ‘computação nas nuvens’ vem do inglês ‘*cloud computing*’, que é a atual tendência para softwares. Alguns anos atrás para você poder acessar um programa, como o *word* (por exemplo), era necessário você instalá-lo no seu computador, pagando uma taxa (em geral alta). Quando falamos em computação nas nuvens, partimos do princípio que você não precisa instalar o *word* no seu computador e poderá acessá-lo pela internet (pagando pelo uso ou um valor fixo mensal – bem abaixo do que quando precisava comprá-lo). Mas isso é possível!? Sim, o *word* pode ser acessado apenas pela internet.

É fácil perceber que os dados desse aplicativo não estão no seu computador, mas em uma rede com acesso à internet, que uma vez que conectado, você poderá desfrutar de todas as suas ferramentas online e acessar seu trabalho de qualquer lugar.

Fonte: RSWA (2015, on-line)<sup>2</sup>.

## CONCEDENDO PERMISSÕES

O comando GRANT é o meio pelo qual é possível conceder privilégios para acessar objetos dentro do Banco de Dados, papéis e públicos. As concessões também permitem conceder privilégios a nível de sistema do Banco de Dados, para usuário e papéis.

As permissões em tabelas podem conceder diferentes tipos de níveis de acesso dentro do Banco de Dados. Esse acesso poderá ser muito específico, como, por exemplo, você pode conceder acesso às colunas específicas de uma tabela. As permissões a nível de sistema permite conceder diferentes funcionalidades a determinados usuários dentro do Banco de Dados, como a capacidade de criar uma tabela ou alterar as configurações de parâmetros de uma sessão de um usuário. Uma vez que um privilégio é atribuído a um usuário, ele entrará em vigor imediatamente.

Segue a sintaxe do GRANT:

```
GRANT ALL ON *.* TO 'usuario'@'seuhost';
```

O comando **REVOKE** revoga direitos de acesso do usuário ou privilégios para os objetos de Banco de Dados. Podemos dizer que esse comando remove os privilégios concedidos pelo comando **GRANT**.

Um determinado usuário pode remover somente os privilégios que foram concedidos diretamente por esse usuário. Se, por exemplo, o usuário José concedeu um privilégio com opção de concessão para o usuário Maria, e o usuário Maria por sua vez concedeu o privilégio para o usuário João, então o usuário José não poderá revogar diretamente o privilégio de João. Em vez disso, o usuário José poderá revogar a opção de concessão do usuário Maria usando a opção CASCADE, para que o privilégio seja, por sua vez, revogado do usuário João. Outro exemplo é o caso em que tanto José quanto Maria concederam o mesmo privilégio a João: nesse caso José poderá revogar sua própria concessão, mas não poderá revogar a concessão feita por Maria e, portanto, João continuará com o privilégio mesmo que José revogue o privilégio.

Segue a sintaxe do REVOKE:

```
REVOKE ALL ON *.* TO 'usuario'@'seuhost';
```

Dessa maneira, aprendemos como conceder e remover os privilégios de um usuário, a nível de um objeto ou mesmo a nível de Banco de Dados. Esse tipo de controle é muito utilizado a fim de garantir a segurança e a integridade das informações no Banco de Dados.



## DTL - LINGUAGEM DE TRANSAÇÃO DE DADOS

O conceito por trás de uma transação faz parte de um paradigma do Banco de Dados relacional. Uma transação completa consiste em um ou mais comandos DML seguidos por um comando COMMIT ou ROLLBACK. Uma sessão começa uma transação a partir do momento que qualquer instrução INSERT, UPDATE ou DELETE é emitida ao Banco de Dados, onde essa transação irá continuar por inúmeros comandos DML, até que a sessão emita uma instrução COMMIT ou ROLLBACK, somente após esses comandos de fato a alteração será persistida no Banco de Dados, e se tornarão visíveis aos demais usuários do Banco de Dados.

- **COMMIT** – Esse tem a função de confirmar os comandos DML aplicados ao Banco de Dados.
- **ROLLBACK** - Faz com que as mudanças nos dados feitos pelos comandos DML existentes, desde o último COMMIT ou ROLLBACK, sejam descartadas.

Digamos que você pediu para excluir dados dentro do Banco de Dados:

```
DELETE FROM pessoa;
```

Quando uma exclusão é efetuada, podemos confirmar essa exclusão com a utilização do comando commit. Tome muito cuidado com essa utilização pois uma confirmação efetuada (commit) se torna irreversível. Esse comando é muito útil e muito utilizado quando temos cenários com vários usuários conectados à mesma base de dados, pois ele trabalha especificamente em cada seção dos usuários. Após o seu uso as modificações se tornam visíveis a todos os demais usuários (BMED, [2016], on-line).<sup>3</sup>

Mas caso não queira que os dados sejam gravados de fato no Banco de Dados, podemos utilizar o **ROLLBACK**, dessa maneira os dados anteriores são “copiados” novamente para o Banco de Dados e retornando assim a sua versão original novamente.



### REFLITA

Tenha em mente que somente dados corretos proporcionarão informações confiáveis para criar estratégias e soluções válidas para a empresa. Afinal, não adianta investir em ferramentas, tecnologias e aplicativos variados, se os dados que estão armazenados não têm qualidade e credibilidade.

## CONSIDERAÇÕES FINAIS

Caro(a) aluno(a), com a finalização do estudo sobre esta unidade, podemos concluir que a prática da SQL é de extrema importância para o amplo entendimento e conhecimento especializado. Durante esta unidade estudamos os principais comandos de um Banco de Dados, para a criação, alteração e remoção das tabelas, comandos para a inserção, alteração e remoção de dados em uma tabela, e também estudamos as permissões dos usuários com a concessão de privilégios e revogação a fim de garantir a segurança e a integridade do nosso Banco de Dados. Vimos que a cada comando DML executado no Banco de Dados é controlado por uma transação, ou seja, por meio dos comandos DTL. Todos esses dados foram trabalhados de uma forma genérica para todos os Banco de Dados que usam como linguagem a SQL.

## ATIVIDADES



1. Quais são os comandos DML?
  - a) *Insert, update, select, delete*
  - b) *Drop, delete, create*
  - c) *Truncate, delete*
  - d) *Insert, update, delete*
  - e) *Rollback, commit.*
2. O comando é responsável por relacionar os atributos desejados no resultado de uma consulta, ou seja, ele permite recuperar os dados de uma tabela do Banco de Dados, esse comando corresponde à operação da álgebra relacional. Estamos falando de?
  - a) *Select*
  - b) *From*
  - c) *Where*
  - d) *and*
  - e) *in*
3. O \_\_\_\_\_ é utilizado para delimitar os dados a serem retornados pela nossa consulta, os filtros a serem aplicados se restringem a linhas utilizando operadores de comparação em um conjunto de campos e valores literários.
  - a. *Select*
  - b. *From*
  - c. *Where*
  - d. *width*
  - e. *view*

## ATIVIDADES



4. Conjunto de comandos dentro da SQL que permite ao desenvolvedor utilizá-lo para a definição das estruturas de dados, contendo instruções as quais permitem a criação, modificação e remoção de tabelas, bem como criação, alteração e remoção de elementos associados às tabelas. (SILBERSCHATZ; KORTH; SUDARSHAN, 1999). Quais são os comandos DDL?
- a) *create , alter, drop*
  - b) *create , alter, drop, update*
  - c) *create , drop*
  - d) *Select, insert, delete*
  - e) *Select, insert, update, delete*
5. **DML** - *Data Manipulation Language* (Linguagem de Manipulação de Dados). Estes comandos são utilizados para realizar inclusões, exclusões e alterações de dados, os quais são utilizados a partir dos comandos INSERT, UPDATE e DELETE. (WATSON; RAMKLASS et al., 2012), comente sobre cada um desses comandos.

# MATERIAL COMPLEMENTAR



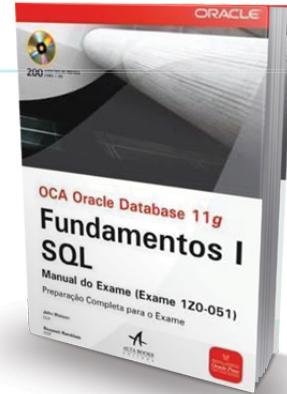
LIVRO

## Fundamentos I SQL (2008)

John Watson e Roopesh Ramklass

**Editora:** Alta Books

**Sinopse:** este livro é um guia de treinamento oficial da Oracle para o exame 1Z0-052. Com mais exercícios, dicas e perguntas de teste. O leitor poderá aprender a criar um Banco de Dados Oracle, configurar uma rede Oracle, administrar a segurança do usuário, fazer backup e recuperação e outros. Este livro inclui um CD com um teste em inglês que simula a experiência do exame.



## REFERÊNCIAS

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados.** 3 ed. São Paulo: Makron Books do Brasil, 1999.

WATSON, J.; RAMKASS, R. **Fundamentos I SQL:** OCA Oracle Database 11g. Rio de Janeiro: Alta Books, 2012.

### Referências on-line:

- 1- Em: <<http://docplayer.com.br/16602595-Linguagem-sql-dml-linguagem-de-manipulacao-de-dados.html>>. Acesso em: 21 nov. 2016.
- 2- Em: <<https://rswa.com.br/2015/07/29/o-que-e-computacao-nas-nuvens/>>. Acesso em: 21 nov. 2016.
- 3- Em: <<http://www.fabiobmed.com.br/comandos-mais-comuns-em-um-banco-de-dados-sql/>>. Acesso em: 21 nov. 2016.



## GABARITO

- 1) D
- 2) A
- 3) C
- 4) A
- 5) O comando **INSERT** é simples, ele é um pedido de inclusão de uma linha em uma tabela. O comando **UPDATE** é utilizado para efetuar alterações nas linhas já existentes no Banco de Dados que possivelmente foram gravadas por meio do comando **INSERT**. **DELETE** é o comando que pode remover uma linha ou um conjunto de linhas da tabela. A quantidade de linhas a serem removidas irá depender da cláusula WHERE.



# MANIPULAÇÃO DE DADOS

UNIDADE



## Objetivos de Aprendizagem

- Identificar na prática a utilidade das teorias aplicadas em relação à linguagem SQL.
- Compreender as atividades básicas do dia a dia de um programador de Banco de Dados.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Extrair dados de uma tabela
- Agrupando a exibição dos dados
- Ordenação na exibição dos dados
- Criando consultas com filtros específicos
- Valores nulos
- Rename
- Consulta utilizando mais de uma tabela
- Consultas com subqueries
- Teste de relações vazias
- Alterando os dados com update
- Removendo os dados com delete
- Rollback e commit
- Truncate e drop



## INTRODUÇÃO

Caro(a) aluno(a)! Nesta unidade vamos nos aprofundar mais sobre os comandos da *Structured Query Language* (SQL), scripts que podemos utilizar por meio de um Sistema de Gerenciamento de Banco de Dados (SGBD). Dessa forma, iremos trabalhar com retorno de dados, consultas e comandos avançados, sendo utilizados para um melhor aproveitamento de nosso Banco de Dados. Conhecemos nas unidades anteriores o conteúdo sobre a linguagem SQL, e por meio dela conhecemos as suas classificações, sendo: a *Data Query Language* (DQL), em português, Linguagem de Consulta de Dados, a *Data Manipulation Language* (DML), em português Linguagem de Manipulação de Dados, a *Data Definition Language* (DDL), em português, Linguagem de Definição de Dados, a *Data Control Language* (DCL), em português, Linguagem de Controle de Dados e a *Data Transaction Language* (DTL), em português, Linguagem de Transação de Dados.

Em Banco de Dados MySQL e para qualquer distribuição de Banco de Dados é extremamente necessário o conhecimento dos conceitos abordados, pois para a aplicabilidade é necessário concretizar as teorias vistas.

Nosso objetivo nesta unidade é demonstrar na prática os conceitos abordados sobre DQL, parte dos comandos DML, pois o comando INSERT já foi abordado na unidade anterior, além dos comandos DTL.

Esperamos que o conteúdo e os assuntos abordados neste livro seja de muito proveito em sua vida profissional, pois o mundo gira em torno da teoria e da prática, ambos necessariamente tem que trabalhar lado a lado, a fim de se obter uma plena excelência sobre o produto desenvolvido. Os melhores sempre são aqueles que têm pleno conhecimento naquilo que fazem.

Bom estudo!



## EXTRAIR DADOS DE UMA TABELA

Caro(a) aluno(a), compreendemos que as informações ficam armazenadas junto ao Banco de Dados. Devemos saber como efetuar a consultas destas informações para assim manipulá-las. Para extrair os dados de uma tabela, ela deve ser consultada pelo SGBD e deverá conter dados. Dessa forma, iremos trabalhar nesta unidade com as informações utilizadas nas unidades anteriores. Primeiramente, vamos relembrar a sintaxe do comando:

```
Select <lista de colunas>
From <tabela>
```

Nossa primeira consulta será na tabela de pessoa, da qual iremos trazer todas as colunas. Na linguagem SQL existe um comando específico para retornar todos os dados da coluna. Esse comando é dado por meio do \* que é uma forma abreviada de dizer “todas as colunas”.

```
Select *
From pessoa;
```

Na Figura 1 podemos observar como a saída de nossa consulta deverá ser.

The screenshot shows a database result grid with the following columns: id\_pessoa, nome, sobrenome, and data\_nascimento. The data in the first row is: 1, João, Pereira, 1983-09-15. The grid has a header row and a footer row with navigation icons.

	id_pessoa	nome	sobrenome	data_nascimento
▶	1	João	Pereira	1983-09-15

Figura 1 - Saída da consulta

Fonte: os autores.

Para que seja possível trabalharmos com os próximos comandos da linguagem SQL será necessário mais de um registro presente na tabela de pessoa. Dessa forma, sugiro que utilize o comando *insert* para efetuar a inserção de mais dados nessa tabela. Dando continuidade em nosso conteúdo sugiro que faça a criação das tabelas que faltam em nossa base de dados seguindo o diagrama:

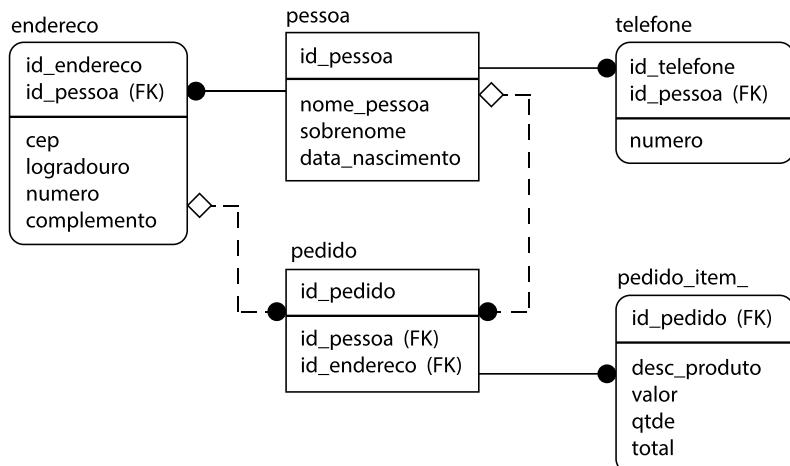


Figura 2 - Tabelas a serem criadas

Fonte: os autores.

Com as tabelas já criadas sugiro que insiram 5 produtos diferentes na tabela de pedido\_item\_. Lembrando que os 5 registros deverão sempre conter o mesmo id\_pessoa. Agora, vamos trabalhar com funções no *select*.

Segundo Silberschatz, Korth e Sudarshan (1999), Funções agregadas são funções que tomam uma coleção (um conjunto ou subconjuntos) de valores de entrada, retornando apenas um valor simples. A linguagem SQL oferece 5 funções agregadas:

- Média (average): **avg**
- Mínimo (minimum): **min**
- Máximo (maximum): **max**
- Total (sum): **sum**
- Contagem (count): **count**

Para a entrada das funções *sum* e *avg* os dados necessariamente precisam ser numéricos. Entretanto, as outras funções podem ser operadas com dados do tipo não numéricos, como as *strings* e seus semelhantes.

Dessa maneira vamos exemplificar a função *sum* com o campo total da tabela *pedido\_item*:

```
select sum(total)  
from unicesumar.pedido_item
```

Nosso resultado deverá apresentar apenas um registro com a soma dos 5 registros desse campo na tabela.

sum(total)
230.00

Figura 3 - Soma dos 5 registros

Fonte: os autores.

Funciona da mesma maneira para a função avg:

```
select avg(total)  
from unicesumar.pedido_item
```

Nesse caso nosso resultado será a média entre os 5 registros da tabela.

Quando desejamos encontrar o número de registros que um *select* irá nos retornar utilizamos a função *count*. A notação para esta função sem SQL é *count(\*)*, como no exemplo a seguir:

```
select count(*)  
from unicesumar.pedido_item
```

O resultado a ser apresentado deverá ser apenas um registro contendo a quantidade dos registros:

	count(*)
▶	5

Figura 4 - Quantidade de registros

Fonte: os autores.

Em nosso exemplo o resultado foi 5, pois só temos 5 registros inseridos na tabela.

A função *min* retorna o menor valor de uma coluna em um grupo de linhas. Podemos utilizá-la para colunas do tipo data ou alfanuméricas. Para saber o preço de venda mais alto do pedido, execute o comando a seguir:

```
select avg(total)  
from unicesumar.pedido_item
```

Já a função *max* retorna o maior valor de uma coluna em um grupo de linhas. Igualmente ao *min*, pode-se utilizá-la para colunas do tipo data ou alfanuméricas. Para saber qual é o produto mais caro do pedido, execute o seguinte comando:

```
select max(total)  
from unicesumar.pedido_item
```

Assim encerramos a principais funções agregadas da linguagem SQL. Essas funções são de uso diário na vida de um programador, pois além de simples, elas são muito úteis.



SAIBA MAIS

Todo Banco de Dados possui alguma linguagem na qual os comandos devem ser enviados por meio de alguma ferramenta *console*. Geralmente essa linguagem conta com elementos padrão do SQL e também com outros elementos adicionais que são específicos do Banco de Dados. De qualquer forma, sempre que um comando não consegue ser executado ou algum outro tipo de problema ocorre, é de responsabilidade do SGBD retornar uma mensagem de erro junto com o código.

Geralmente essa mensagem de erro está em inglês, apesar de existirem alguns trabalhos de tradução para o português. Além disso, todas as mensagens de erro devem estar devidamente descritas e apresentadas na documentação oficial do Banco de Dados.

Para ler o artigo na íntegra acesse:

<<http://imasters.com.br/banco-de-dados/quais=sao-os-erros-mais-comuns-em-bancos-de-dados/?trace-1519021197&source=single>>

Fonte: Pichiliani (2013, on-line)<sup>1</sup>.



## AGRUPANDO A EXIBIÇÃO DOS DADOS

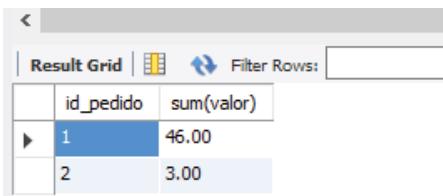
Com a utilização do *group by*, é possível efetuar o agrupamento de diversos registros baseados em uma ou mais colunas de uma tabela. Por exemplo, os produtos da tabela pedido\_item podem ser agrupados pelo **valor** (maior valor, menor valor), pelo **qtde** etc. Importante frisar que a cláusula *group by* é comumente em conjunto com as funções agregadas.

O *group by* é responsável por determinar em quais grupos devem ser colocadas as linhas de saída. Caso nosso *select* tenha funções agregadas, a cláusula *group by* realiza um cálculo a fim de chegar ao valor sumário para cada um dos grupos.

Para colocarmos essa cláusula devemos estar diante de uma das seguintes situações: ou a expressão *group by* deve ser correspondente à expressão da lista de seleção ou cada uma das colunas presentes em uma expressão não agregada na lista de seleção deve ser adicionada à lista de *group by*. Observe o exemplo a seguir:

```
select id_pedido,
       sum(valor)
  from unicesumar.pedido_item
 group by id_pedido
```

O *select* nos traz a soma da coluna valor agrupador pelo id\_pedido, conforme explicação anterior. Para exemplificar esta consulta foi inserido mais um pedido na tabela de pedido e mais alguns itens relacionados com o segundo pedido.



	id_pedido	sum(valor)
▶	1	46.00
	2	3.00

Figura 5 - Novo pedido

Fonte: os autores.

## ORDENAÇÃO NA EXIBIÇÃO DOS DADOS

Segundo Silberschatz, Korth e Sudarshan (1999), a linguagem SQL oferece ao usuário um controle sobre a ordem que os dados são retornados para o usuário. A Cláusula *order by* faz com que os registros do resultado de uma consulta apareçam na ordem classificada. Para listar por ordem alfabética todos os produtos da tabela de produto\_item devemos escrever da seguinte maneira:

```
Select desc_produto, id_pedido  
From pedido_item  
Order by desc_produto
```



Por padrão a cláusula *order by* lista os resultados em forma crescente. Para que seja possível especificar a ordem de retorno, podemos especificar *desc* para decrescente ou *asc* para a ordem crescente dos dados. Além disso, a ordenação pode ser feita por mais de 1 campo da consulta. Supondo que precisamos ordenar uma consulta por mais de um campo:

```
Select *
From pedido_item
Order by desc_pedido, valor
```

Dessa forma, primeiro nossa consulta irá ser ordenada pela descrição do produto e, posteriormente, irá ordenar o segundo campo, sempre respeitando a ordenação anterior. Vale lembrar que como a classificação de um grande número de registros pode ser demorada, esse tipo de classificação deverá ser feita somente quando necessário.

## CRIANDO CONSULTAS COM FILTROS ESPECÍFICOS

Vamos ilustrar o uso da cláusula *where* na SQL. Consideremos então a seguinte consulta, “encontre todos os registros no pedido os quais o valor é maior que 20”



```
Select *  
From pedido_item  
Where valor > 20
```

Na linguagem SQL utilizamos *and*, *or* e *not* ao invés de utilizarmos notações matemáticas  $\wedge$ ,  $\vee$  e  $\neg$  na cláusula *where*. Os operandos dos conectivos lógicos podem ser expressões envolvendo os seguintes operadores de comparação  $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $=$  e  $\neq$ . Essa linguagem permite operadores que comparam *strings*, expressões aritméticas além de comparativos entre datas.

A linguagem SQL possui um operador de comparação *between*, que tem o intuito de simplificar a cláusula *where*. Seu intuito é encontrar um valor que seja menor ou igual a algum valor e maior ou igual a um outro valor. No caso, se quisermos encontrar os produtos que estão entre 10 e 20, devemos escrever a consulta da forma apresentada a seguir:

```
Select *  
From pedido_item  
Where valor between 10 and 20
```

Ao invés de:

```
Select *  
From pedido_item  
Where valor >= 10 and valor <= 20
```

Da mesma forma também é possível utilizar o comando de comparação *not between*.

## VALORES NULOS

Podemos realizar o uso de valores nulos para denotar a ausência de informações nos registros. Na SQL utiliza-se a palavra *null* para testar a presença de um valor nulo, por exemplo, para buscar os registros que não possuem valor, devemos escrever a seguinte consulta:

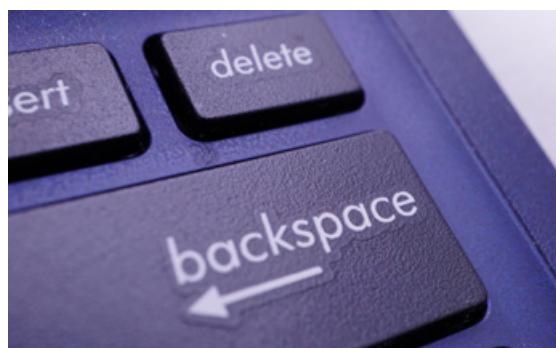
```
Select *  
From pedido_item  
Where valor is null
```



Também podemos utilizar o predicado *is not null* que testa a ausência de um valor nulo no campo.

## RENAME

Um grande mecanismo da linguagem SQL, muito utilizado por todos, é a função de renomear um campo ou tabela da consulta, empregando a utilização da cláusula AS da seguinte forma:



Nome\_do\_campo\_ou\_tabela **as** nome\_novo

A utilização dessa cláusula se dá tanto no *select*, quanto no *from* das consultas. Consideremos a seguinte consulta:

```
Select pedido_item.desc_produto,  
      Pedido_item.valor,  
      pedido_item.qtde  
From  pedido_item
```

Agora utilizando a função de rename.

```
Select pd.desc_produto,  
      pd.valor,  
      pd.qtde  
From  pedido_item as pd
```

Dessa maneira, rebatizam o nome da tabela, para que a consulta fique mais simples.

A utilização na cláusula *from* é logo depois do nome da tabela a qual se deseja a associação, com a palavra **as** entre eles, como no exemplo anterior. Vale lembrar que a utilização do **as** é opcional, pois o próprio *sgbd* subentende essa utilização. Assim como em muitos livros, aqui iremos tratar esse *rename* como a criação de um **alias** para as tabelas ou colunas.

## CONSULTA UTILIZANDO MAIS DE UMA TABELA

Para que seja possível efetuar uma consulta utilizando mais de uma tabela, é necessário trabalharmos com a cláusula *from* de nosso *select*. Veja no exemplo a seguir que estamos acrescentando uma tabela no *from*, a tabela “Tabela2”, lembrando que para essa prática é necessário “ligarmos” as tabelas com a cláusula *where* em suas respectivas chaves primárias e estrangeiras (ANTUNES, on-line, [2016])1.



```
SELECT
    Tabela1.coluna1,
    Tabela1.coluna2,
    Tabela2.coluna1,
    Tabela2.coluna2
FROM Tabela1, Tabela2
WHERE Tabela1.chave_primaria = Tabela2.chave_estrangeira
```

Vale lembrar que a utilização de alias é opcional antes do nome das colunas. Contudo, essa é uma prática muito utilizada, facilitando assim o entendimento e uma possível manutenção no código.

A utilização do alias antes do nome do campo se torna obrigatório quando temos o mesmo campo em mais de uma tabela de nossa consulta. Geralmente, isso acontece muito com as chaves primárias e estrangeiras, sendo assim neste caso é obrigatório que se indique de qual tabela deverá retornar o dado. Considere que informar em qual tabela está a coluna facilita o trabalho do Banco de Dados. Essa prática leva a maior agilidade na recuperação da informação. A união regular se dá pela união na cláusula *where* da chave primária com a chave estrangeira das tabelas.

## CONSULTAS COM SUBQUERIES

Segundo Silberschatz, Korth e Sudarshan. (1999), a SQL fornece um mecanismo para aninhar subconsultas. Uma subconsulta é uma expressão *select - from - where* que é utilizada dentro de outra consulta. Esse uso é muito comum para realizar teste de participação de conjuntos, fazer comparações e determinar a cardinalidade dos conjuntos.

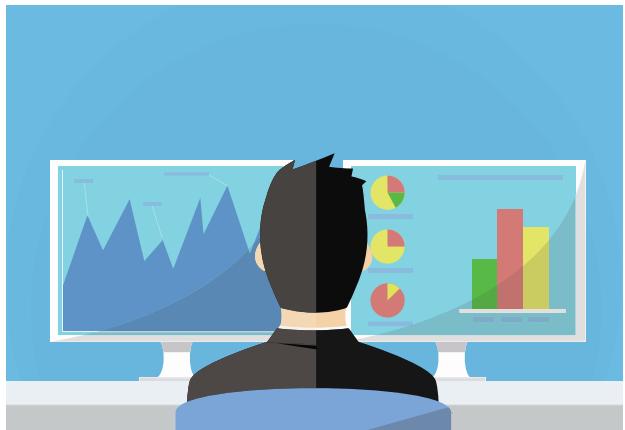
O conectivo IN testa um conjunto de valores produzidos pelo *select* na cláusula *where*, ou ainda podemos trabalhar com o NOT IN que testa a ausência de um conjunto de valores.

Para exemplificar essa utilização, vamos pensar em um *select* que retorna todos os pedido dos clientes cadastrados em nossa base. Primeiro, vamos encontrar os identificadores da tabela de pessoa:

```
Select id_pessoa  
From pessoa
```

Depois de encontrar os dados da pessoa, vamos aninhar essa consulta com a consulta dos pedidos. Vale lembrar que a subconsulta sempre deve estar entre parênteses.

```
Select *  
From pedido  
Where id_pessoa in (Select id_pessoa  
From pessoa)
```



Selects que incluem uma subconsulta normalmente têm um destes formatos:

- WHERE expressão [NOT] IN (subconsulta)
- WHERE expressão operador de comparação [ANY | ALL] (subconsulta)
- WHERE [NOT] EXISTS (subconsulta)

## TESTE DE RELAÇÕES VAZIAS

A função *Exists* e *Not Exists* da SQL é usada para verificar se o resultado de uma consulta aninhada correlacionada é vazio (não contém nenhum registro) ou não. A construção do *exists* retorna um valor **true** se a subconsulta não é vazia. Pensando dessa maneira, vamos escrever um *select* que nos traga somente os dados das pessoas que possuem pedido.



```
Select pes.*  
From pessoa pes  
Where exists (select ped.*  
From pedido ped  
Where ped.id_pessoa = pes.id_pessoa)
```

Da mesma maneira, podemos testar a existência dos registros utilizando o *not exists*. De acordo com o exemplo, se utilizamos o *not exists* iremos trazer os dados de todas as pessoas que não possuem nenhum pedido.

```
Select pes.*  
From pessoa pes  
Where not exists (select ped.*  
                    From pedido ped  
                    Where ped.id_pessoa = pes.id_pessoa)
```

## ALTERANDO OS DADOS COM UPDATE

Em muitas situações temos que alterar um dado de uma tabela, sem alterar todos os seus valores. A instrução UPDATE deve ser utilizada nesse caso. Bem parecida com a instrução de *insert* e *delete*, podem ser selecionados os campos que devem ser atualizados.

Supondo que desejamos alterar o valor dos produtos em 10%, deveremos escrever o seguinte comando:

```
Update pedido_item  
Set valor = valor * 1.1
```



O comando anterior é aplicado uma vez em cada registro da tabela de pedido\_item. Caso desejarmos, por exemplo, aumentar em 10% o valor dos produtos que custam mais de R\$10, deveríamos escrever o seguinte comando:

```
Update pedido_item  
Set valor = valor * 1.1  
Where valor >=10
```

A cláusula *where* da instrução *update* é muito semelhante a instrução *where* do *select*, incluindo *select* aninhado. Como no *insert* e *delete*, um *select* pode referenciar uma relação a ser atualizada do *update*. Pensando dessa forma, vamos escrever um *update* que atualiza o valor dos produtos que estão acima da média em 10%.

```
Update pedido_item  
Set valor = valor * 1.1  
Where valor > (select avg(valor)  
From pedido_item)
```

O update fornece uma instrução chamada **case** que nos dá a opção de fazer duas atualizações com uma única instrução *update*. Em uma forma geral a instrução *case* é a seguinte:

```
Case  
When predicado1 then  
    Resultado1  
When predicado12 then  
    Resultado2  
Else  
    resultado0  
End
```

Para atualizarmos o valor dos produtos menores que R\$10,00 em 10% e os produtos maiores que R\$10,00 em 5%, devemos escrever o seguinte comando:

```
Update pedido_item
Set valor = case
    When valor <=10 then
        Valor = valor * 1,1
    Else
        Valor = valor * 1,05
End
```

## REMOVENDO OS DADOS COM DELETE

Uma remoção de dados de uma tabela, às vezes, é expressa do mesmo modo que uma consulta, podendo serem removidos apenas registros completos, não podendo assim remover o conteúdo de um campo. Podemos expressar uma remoção da seguinte maneira:



```
Delete from nome_da_tabela
Where condição
```

Vale lembrar que o comando *delete* trabalha apenas com uma relação, caso seja necessário remover os dados de diversas tabelas é necessário utilizar um comando de *delete* para cada tabela. A cláusula *where* pode ser tão complexa quanto a cláusula *where* de um *select*, porém, pensando em simplicidade, podemos também

executar o comando *delete* sem o uso da cláusula *where* e assim, todos os registros da tabela em questão serão apagados. Exemplo:

```
Delete from pedido_item
```

No exemplo anterior, estamos apagando todos os itens registrados na tabela de *pedido\_item*.

Apresentamos aqui uma série de exemplos de remoção utilizando a SQL:  
Delete os registros desde que o valor seja maior que 10:

```
Delete from pedido_item where valor > 10
```

Apaga os registros que tenham o valor de produto acima da média:

```
Delete from pedido_item where valor > (select avg(valor) from pedido_item)
```

## ROLLBACK E COMMIT

Segundo Silberschatz, Korth e Sudarshan (1999), uma transação consiste em uma sequência de instruções de consulta ou de atualização. O padrão SQL especifica que uma transação inicia implicitamente quando uma instrução SQL é executada, em que uma das seguintes instruções precisam finalizar a transação:

- **Commit:** confirma a transação atual. Ou seja, torna as atualizações realizadas pela transação permanentes no Banco de Dados. Após a confirmação da transação, uma nova transação é iniciada.
- **Rollback:** faz com que a transação atual seja revertida: ou seja, ele desfaz todas as atualizações realizadas pela instrução SQL na transação. Portanto, o estado do Banco de Dados é restaurado para como era antes da primeira instrução da transação ser executada.

## TRUNCATE E DROP

Para que seja possível remover uma tabela de um Banco de Dados utilizamos o comando *drop table*. O comando *drop table* exclui do Banco de Dados a tabela e todas as informações nela contida. Esse comando é muito simples, segue a sintaxe:

```
Drop table nome_da_tabela
```

O comando TRUNCATE é responsável por limpar os registros de uma tabela e fará isso de uma forma mais rápida que o comando DELETE. Esse comando é mais rápido que o comando *delete* porque ele não faz uma cópia dos dados. Assim, não se tem o comando de *rollback*. O comando TRUNCATE é um comando DDL, enquanto o DELETE é um comando DML. Observe as principais diferenças entre o DELETE e o TRUNCATE:

1. TRUNCATE é um comando de Linguagem de Definição de Dados, enquanto DELETE é de Manipulação de Dados.
2. O Comando TRUNCATE não conta com a função *RollBack* ao contrário do DELETE.
3. Uma TRIGGER não é disparada quando utilizamos o TRUNCATE, com o comando DELETE, se existir ela será disparada.
4. O comando TRUNCATE apaga todos os dados da tabela, já o DELETE podemos associá-lo a condições (cláusula WHERE) (LEITÃO, 2015, on-line)<sup>3</sup>.

Pensando dessa forma utilizamos o comando *drop*, para apagar as tabelas do Banco de Dados juntamente com os seus respectivos conteúdos. Já o comando *truncate*, por sua vez, para limpar os dados de uma tabela.

REFLITA



A manipulação das informações em um Banco de Dados, ocorre somente com a linguagem SQL ou podemos utilizar programas que nos auxiliem nesse processo?

## CONSIDERAÇÕES FINAIS

Prezado(a) aluno(a)! Após conhecermos os comandos básicos da linguagem SQL, sendo os de Linguagem de Manipulação de Dados (DML), Linguagem de Definição de Dados (DDL) e Linguagem de Controle de Dados (DCL), entre outros apresentados na unidade anterior, apresentamos por meio desta unidade alguns conceitos e comandos mais avançados sobre o SQL.

Foram apresentados comandos e formas mais avançadas de seleção de dados em uma tabela. Alguns que facilitam a manipulação da informação e ajudam a refinar nossas buscas, tornando a consulta mais eficaz. Por meio dessas consultas, a manipulação de dados fica mais clara, podemos citar como exemplo os comando de contagem, soma, ordenação de apresentação dos dados (crescente ou decrescente) e a cláusula *Where*, está sendo fundamental em várias consultas ao Banco de Dados.

Ao falarmos de seleção de dados nos deparamos em casos que precisamos manipular mais do que uma tabela. Trabalhamos também nesta unidade essa questão que pode nos ajudar em nosso processo de desenvolvimento de software. Nesse sentido, aprendemos duas formas de efetuar por meio de uma consulta em várias tabelas ou *subqueries*.

Entendemos que em um Banco de Dados as informações podem ser modificadas a qualquer momento. Pensando nisto conhecemos algumas formas de efetuar a alteração dos dados por meio dos comando UPDATE, além de entendermos a função dos comandos ROLLBACK e COMMIT, esses sendo úteis em casos de situação efetuadas de modo errado no Banco de Dados.

Por fim, conhecemos os comandos TRUNCATE, DELETE e DROP. Todos têm o mesmo princípio, limpar os dados da tabela, cada um com sua particularidade, comandos que podem ser perigosos se não forem manipulados de modo adequado.

Chegamos ao fim de mais uma unidade, espero que tenha contribuído de forma significativa em sua aprendizagem, prezado aluno(a), um forte abraço e até a próxima.

## ATIVIDADES



1. Com a utilização do *group by*, é possível efetuar o agrupamento de diversos registros baseados em uma ou mais colunas de uma tabela. Dessa maneira dê o exemplo de um *select* utilizando o *group by* e o porquê de sua utilização.
2. Sabemos que o *truncate* é responsável por limpar os registros de uma tabela, e que ele é mais rápido que o comando *delete*. Esse comando é mais rápido por não se tratar de um comando DML. Dessa maneira, cite as principais diferenças entre os comandos *DELETE* e *TRUNCATE*.
3. Uma subconsulta é uma expressão *select - from - where* que é utilizada dentro de outra consulta. Esse uso é muito comum para se realizar teste de participação de conjuntos, fazer comparações e determinar a cardinalidade dos conjuntos. O conectivo *IN* testa um conjunto de valores produzidos pelo *select* na cláusula *where*, ou ainda podemos trabalhar com o *NOT IN* que testa a ausência de um conjunto de valores. Exemplifique a utilização de *subqueries* de acordo com alguma tabela de nosso livro.
4. A linguagem SQL possui um operador de comparação *between* que tem o intuito de simplificar a cláusula *where*. O intuito é encontrar um valor que seja menor ou igual a algum valor e maior ou igual a um outro valor. Pensando dessa forma escreva um *select* que encontre os produtos que estão com valor entre 3 e 8.
5. Segundo Silberschatz, Korth e Sudarshan (1999), Funções agregadas são funções que tomam uma coleção (um conjunto ou subconjuntos) de valores de entrada, retornando apenas um valor simples. Descreva e relacione essas funções.

# MATERIAL COMPLEMENTAR



LIVRO

## Sistemas de Banco de Dados (2012)

Virgínia M. Cardoso e Giselle Cristina Cardoso

**Editora:** Saraiva

**Sinopse:** em texto introdutório, as autoras apresentam tópicos que levam o aluno a conhecer o universo do Banco de Dados, apresentando como projetar, construir e popular um Banco de Dados por meio de exemplos didáticos e práticos. De forma a facilitar o aprendizado, este texto proporciona ferramentas para que o aluno seja capaz de desenvolver de forma independente um Banco de Dados com armazenamento seguro e que ofereça uma pesquisa rápida e eficaz.





## REFERÊNCIAS

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados.** 3 ed. São Paulo: Makron Books do Brasil, 1999

### Referências on-line:

- 1- Em: <<http://imasters.com.br/banco-de-dados/quais-sao-os-erros-mais-comuns-em-bancos-de-dados/?trace=1519021197&source=single>>. Acesso em: 22 nov. 2016.
- 2- Em: <<http://www.devmedia.com.br/introducao-ao-sql-pesquisa-em-multiplas-tabelas/17006>>. Acesso em: 22 nov. 2016.
- 3- Em: <<http://www.profissionaloracle.com.br/gpo/servicos/easyblog/entry/2015/08/11/diferenca-entre-os-comandos-truncate-delete-e-drop?tmpl=component&type=raw>>. Acesso em: 22 nov. 2016.



# GABARITO

1. Utilizado para efetuar o agrupamento pelo id\_produto da tabela de pedido

```
select id_pedido,  
       sum(valor)  
  from unicesumar_pedido_item
```

2. As principais diferenças são:

TRUNCATE é um comando DDL enquanto DELETE é um comando DML.

TRUNCATE é muito mais rápido do que o DELETE.

Não existe Rollback para o comando TRUNCATE, mas para o DELETE sim. O comando TRUNCATE re move o registro permanente.

Em caso de TRUNCATE, a TRIGGER não é disparada, mas no caso do comando DELETE, existindo TRIGGER para deleção, ela é disparada.

Você não pode usar condições (cláusula WHERE) com o comando TRUNCATE. Mas com o comando DELETE, você pode escrever usando condições (cláusula WHERE).

- 3.

```
Select *  
From pessoa  
Where id_pessoa in (Select id_pessoa  
                      From pedido)
```

Nesse select estamos retornando somente as pessoas que possuem pedido no Banco de Dados.



## GABARITO

4.

```
Select *  
From pedido_item  
Where valor between 3 and 8
```

5. Média (average): **avg**

- **Min** - Traz o menor valor da coluna.
- **Max** - Traz o maior valor da coluna.
- **Sum** - Soma os valores da coluna.
- **Count** - Retorna a quantidade de registros da consulta em questão.
- **Avg** - Faz a média dos registros para esta coluna retornados em um select.





# PROGRAMAÇÃO EM SQL

UNIDADE

IV

## Objetivos de Aprendizagem

- Conhecer algumas diferenças da ferramenta utilizada.
- Entender a programação de Store Procedures.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Visão Geral sobre PL/SQL
- Procedures
- Functions
- Packages



## INTRODUÇÃO

Olá caro(a) aluno(a)! Compreender um produto tão grandioso quanto um Banco de Dados é conseguir obter uma noção de como efetivamente ele deve funcionar, ter o conhecimento dos detalhes intrínsecos da formação do sistema e o seu gerenciamento.

O objetivo é ter uma base abrangente sobre os conceitos e tecnologias que formam os fundamentos de um Servidor de Banco de Dados utilizando os seus melhores recursos.

A ferramenta utilizada será o Oracle. A faixa de utilizadores passa por usuários e desenvolvedores, e até Administradores de Banco de Dados (DBA), sendo uma das opções mais populares e eficazes do mercado.

A compreensão básica sobre o produto, deve proporcionar a ligação dos pontos para a utilização do volumoso conjunto de características e documentação do Oracle, bem como os muitos livros e publicações que descrevem essa base de dados.

A empresa, antes pouco conhecida, sendo apenas uma a mais entre as concorrentes, levou anos em desenvolvimento de suas soluções para o mercado de gerenciadores de bancos de dados até se tornar a líder mundial no segmento, oferecendo um produto com níveis cada vez melhores em escalabilidade, funcionalidade e gerenciamento dos dados.

Vamos abordar algumas diferenças entre o Oracle e o MySQL para que possamos equalizar as funcionalidades entre eles e compreender as funcionalidades dos comandos de programação e estruturação de códigos e suas particularidades.

Ótimo estudo!



Figura 1 - O poder da linguagem PL/SQL

## Visão Geral Sobre PL/SQL

O PL/SQL (*Procedural Language/SQL*) é uma extensão para o (*Structured Query Language*) SQL, incorporando várias facilidades das linguagens de programação existentes. O PL/SQL permite utilizar comandos para manipulação de dados e consultas em blocos de programação estruturados, fazendo do PL/SQL uma poderosa linguagem de processamento de transações.

O objetivo desta visão geral sobre PL/SQL é distinguir entre um bloco PL/SQL anônimo e nomeado, descrever subprogramas, listar os benefícios na utilização de subprogramas e descrever onde um subprograma pode ser chamado.

Para iniciar esta abordagem, é fundamental conhecer um subprograma. Ele é um bloco padrão de PL/SQL nomeado que aceita parâmetros e pode ser chamado de um ambiente. São dois os tipos de subprogramas: *functions* e *procedures*.

Um subprograma é modular, pode ser reutilizável, extensível e gerenciável, fornece a maior segurança aos dados, melhorando a performance do sistema, a corretude do código e a integridade dos dados, oferecendo mais clareza ao código. São blocos nomeados. Estes podem ser declarados como uma *procedure* ou função. Em alguns casos, pode retornar valores quando executados.

## ESTRUTURA DE BLOCO

PL/SQL é uma linguagem estruturada em blocos. Variáveis podem ser definidas em blocos que representam seu escopo de utilização, e erros podem ser tratados dentro do bloco.

Controles de fluxo por meio de comandos IF, THEN, ELSE e ELSIF controlam os desvios do fluxo de execução em tempo real.

A portabilidade e integração dos códigos PL/SQL é garantida em qualquer ambiente executando uma base de dados Oracle ou que o suportem.

Em termos de desempenho, codificação PL/SQL pode proporcionar bons níveis de performance em certos ambientes, favorecendo seu uso (BAC, [2016], on-line)<sup>1</sup>.

## ESTRUTURA DO PL/SQL

As unidades feitas em PL/SQL podem ser definidas em um ou mais blocos. Esses blocos são como componentes que podem estar, inclusive, embutidos em outro bloco como se fossem subrotinas.

Um exemplo de estrutura para esses blocos inicia com um trecho de declaração, seguido pelo trecho principal de comandos do bloco, e finalmente, a parte de comandos para tratamento de exceções como indicado a seguir (BAC, [2016], on-line)<sup>1</sup>:

```
DECLARE (Opcional)
        Variáveis, cursores, exceções
BEGIN (Obrigatório)
        -Declarações SQL
        -Declarações PL/SQL
EXCEPTION (Opcional)
        Manusear ações a serem tomadas quando ocorrerem erros
        pré-definidos
END; (Obrigatório)
```

**IMPORTANTE:** os comandos DECLARE, BEGIN e EXCEPTION não são precedidas de ponto e vírgula. Todos os outros comandos PL/SQL, incluindo o END exigem ponto e vírgula.

No geral, um bloco pode ser anônimo declarado em qualquer ponto do código para ser executado no PL/SQL *Engine*, ou nomeado (subprograma) representando subprogramas e podendo representar *Procedures* ou *Functions* (com retorno de valores) (BAC, [2016], on-line)1.

## SINTAXE BÁSICA DO PL/SQL

Sabendo que o PL/SQL é uma extensão do SQL, geralmente as regras de sintaxe que são aplicadas no SQL são aplicadas no PL/SQL. Ao trabalharmos com a Linguagem SQL, temos que nos atentar sobre alguns comandos reservados, exemplos: DATABASE, SHOW, USE, VARCHAR, entre outros. Eles podem ser utilizados em alguns casos especiais utilizando entre aspas duplas (“USE”), comentários podem estar entre /\* e \*/, e atribuições são feitas com dois pontos e sinal de igual (BAC, [2016], on-line)1.

O controle do fluxo dos comandos pode utilizar as instruções básicas de linguagens de programação:

- IF - Efetua um controle de ações baseado em condições.
- GO TO - Efetua um desvio incondicional para um ponto determinado no programa.
- LOOP - END LOOP; - Para repetições de ações sem uma condição imposta.
- FOR - Controla as repetições de ações utilizando um contador.
- WHILE - Controla as repetições de ações baseado em condições.
- EXIT - Termina uma repetição.



Figura 2 - Sintaxe para Banco de Dados

## ATRIBUTOS UTILIZADOS

Os atributos, utilizados para declarar um registro com base em uma coluna, ou em uma coleção de colunas, são muito úteis para a leitura e recuperação de informações nas estruturas PL/SQL. São identificados na seção DECLARE.

São eles:

- **%TYPE** - é utilizado para declarar um registro baseado numa coluna de uma tabela ou visão.

Exemplo:

```
DECLARE
    X_nome Y.nome%TYPE;
BEGIN
    SELECT nome INTO X_nome FROM Y WHERE nome = 1234;
    ...
END;
```

- **%ROWTYPE** - é utilizado para declarar um registro baseado numa coleção de colunas de uma tabela ou visão. Os campos dentro dos registros receberão seus tipos de dados das colunas referenciadas.

```
DECLARE
    id_funcionario Y.nome%ROWTYPE;
BEGIN
    SELECT * INTO id_funcionario FROM Y WHERE reg_funcionario = 100000;
    ...
END;
```



## REFLITA

Desde a versão 7 o Oracle possui dois otimizadores por Regra (Ruled Based Optimizer - RBO) e por Custo (cost Based Optimizer - CBO). Você conhece os otimizadores? Conhece os benefícios deles?

## PROCEDURES

*Procedure* é um subprograma que executa uma ação. Pode ser armazenada no Banco de Dados, com um objeto pertencente ao esquema, para execuções futuras. A segurança de utilização da procedure é gerenciada pelo próprio Banco de Dados.

Para criar uma nova procedure, utilizar o comando CREATE PROCEDURE. Com uma lista de argumentos, e definindo as ações que serão executadas pelo bloco PL/SQL, a sintaxe é:

```
CREATE [OR REPLACE] PROCEDURE nome_procedimento
  [(parametro1 [modo] tipo1,
    Parametro2 [modo2] tipo2,
    . . . ) ]
IS | AS
  PL/SQL Block ;
```



Figura 3 - Apresentação de procedure

## PARÂMETROS

Existem dois tipos de parâmetros para a execução das *procedures*, são Formais e Reais.

Os parâmetros formais são variáveis declaradas na lista de parâmetros da especificação de um subprograma. Por exemplo:



Figura 4 - Os parâmetros são o caminho

```
CREATE PROCEDURE procedimento (id number)
...
END procedimento;
```

Os parâmetros Reais são variáveis ou expressões utilizadas na chamada de um subprograma, por exemplo: procedimento (7788).

A apresentação dos Parâmetros pode ser do tipo entrada, saída ou entrada e saída simultâneos.

Para criar *procedures* com parâmetros, os detalhes podem ser observados na Tabela 1.

Tabela 1 - Criando procedures com parâmetros

IN	OUT	IN OUT
Modo padrão.	Deve ser especificado.	Deve ser especificado.
Valores de entrada para subprogramas.	Valores retornados para o ambiente chamador dos subprogramas.	Valores de entrada para o subprograma e retornados para o ambiente chamador.
Parâmetros formais, constantes.	Variável não inicializada.	Variável inicializada.
Pode ser atribuído, mas não retorna um valor.	Não pode ser atribuído, mas retorna um valor	Pode atribuir e retornar um valor.

Parâmetros reais podem ser constantes, expressões, literais e variáveis.	Deve ser uma variável.	Deve ser uma variável.
--	------------------------	------------------------

Fonte: os autores.

Observe a seguir um exemplo de um parâmetro IN, criando uma *procedure* com o argumento IN, armazenando todas as informações de um novo empregado:

```
CREATE TABLE LOG_TABLE ( USUARIO CHAR(20) NOT NULL,
                        DATA DATE DEFAULT SYSDATE );
CREATE OR REPLACE PROCEDURE log_execution IS
BEGIN
  INSERT INTO LOG_TABLE (USUARIO, DATA)
    VALUES (USER, SYSDATE);
END;
```

Agora, caro (a) aluno (a), observe um exemplo de um parâmetro OUT, retornando um valor da procedure para o ambiente chamador utilizando o argumento OUT, devolvendo informações sobre um funcionário:

```
CREATE OR REPLACE PROCEDURE PESQUISA
  (ID IN REGISTRO%TYPE,
   NOME      OUT EMP.ENAME%TYPE,
   SAL       OUT EMP.SAL%TYPE,
   COM       OUT EMP.COMM%TYPE)
IS
BEGIN
  SELECT ENAME, SAL, COMM
    INTO NOME,     SAL,   COM
   FROM EMP
  WHERE EMPNO = EMP_ID;
END PESQUISA_EMP;
```

Agora, caro (a) aluno (a), execute da seguinte forma:

```
SQL> VARIABLE E_NOME VARCHAR2(30)
SQL> VARIABLE E_SAL    NUMBER
SQL> VARIABLE E_COM    NUMBER
SQL> EXECUTE QUERY_EMP(7788, :E_NOME, :E_SAL, :E_COM);
```

A partir disso, retorna a mensagem: “PL/SQL procedure successfully completed.” O passo seguinte é a apresentação do resultado da *Procedure* com parâmetro OUT, conforme o código apresentado:

```
PRINT E_SAL
```

O resultado é apresentado da seguinte forma:

```
E_SAL
```

```
-----
```

```
3000
```

Por último, veremos o exemplo de um parâmetro IN OUT, passando um valor do ambiente chamador para a *procedure*, e a *procedure* retornando um valor para o ambiente chamador. Na sequência, é apresentado o código da criação desta *procedure*:

```
CREATE OR REPLACE PROCEDURE FORMATA_CNPJ
(P_CNPJ IN OU VARCHAR2)
IS
BEGIN
    P_CNPJ := SUBSTR(P_CNPJ, 1,2) || '.' || |
    P_CNPJ := SUBSTR(P_CNPJ, 3,3) || '.' || |
    P_CNPJ := SUBSTR(P_CNPJ, 6,3) || '/' || |
    P_CNPJ := SUBSTR(P_CNPJ, 9,4) || '-' || |
    P_CNPJ := SUBSTR(P_CNPJ, 13,2);
END FORMATA_CNPJ;
```

A mensagem “**Procedure created.**” é apresentada, e a variável deve ser criada e carregada com o valor para a formatação, observe a seguir:

```
VARIABLE CNPJ VARCAHAR2(20)
BEGIN
    :CNPJ := '12345678901234';
END;

EXECUTE FORMATA_CNPJ(: CNPJ);
```

A mensagem “**PL/SQL procedure successfully completed.**” é apresentada, e para visualizar o CNPJ formatado, veja a seguir:

PRINT CNPJ

O CNPJ será apresentado formatado da seguinte forma:

CNPJ

-----  
12.345.678/9012-34

O método de passagem dos parâmetros também exige atenção, e pode ocorrer de três formas:

- **Posicional:** lista dos parâmetros reais na mesma ordem dos parâmetros formais.
- **Nomeado:** lista dos parâmetros em ordem arbitrária, porém relacionados pelos nomes.
- **Combinado:** alguns parâmetros posicionais e outros nomeados. Os valores padrões podem ser informados diretamente no código, e também pode ser observado um modelo de passagem de parâmetros, conforme o código. Observe a seguir:

```

CREATE OR REPLACE PROCEDURE add_dept
    (p_name    IN departments.department_name%TYPE
     DEFAULT 'unknown',
     p_loc      IN departments.location_id%TYPE
     DEPAULT 1700)
IS
BEGIN
    INSERT INTO departments(department_id,
                           Department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name,
            p_loc);
END add_dept;

BEGIN
    add_dept;
    add_dept ('TRAINING', 2500);
    add_dept ( p_loc => 2400, p_name => 'EDUCA-
TION');
    add_dept ( p_loc => 1200);
END;

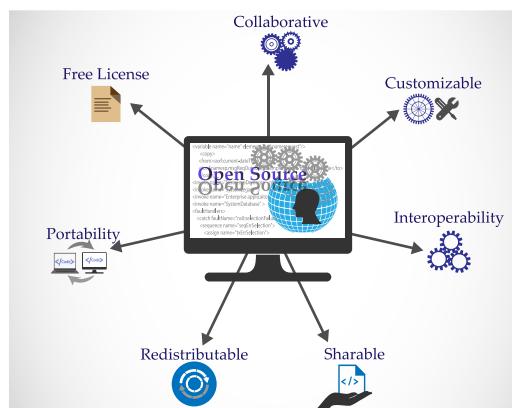
```

## SUBPROGRAMAS

Conforme a necessidade de trabalhar com eventos sincronizados, ou de outra ordem, as *procedures* podem trabalhar com subprogramas automatizando e realizando esses eventos.

Um subprograma é de fácil manutenção, melhora a segurança e integridade dos dados, melhora a performance e também o código.

A gerência dos subprogramas, que são procedimentos e funções armazenadas no



Banco de Dados, acontece por meio dos comandos de criação e exclusão, e são respectivamente:

```
CREATE OR REPLACE PROCEDURE/FUNCTION  
DROP PROCEDURE/FUNCTION
```

Um exemplo interessante de um subprograma é da exclusão de um registro, e instantaneamente, a inclusão de um *log* de exclusão, conforme apresentado no código que segue com o subprograma em destaque:

```
CREATE OR REPLACE PROCEDURE leave_emp2  
    (p_id    IN employees.employee_id%TYPE)  
IS  
    PROCEDURE log_exec  
    IS  
    BEGIN  
        INSERT INTO log_table (user_id, log_date)  
        VALUES (USER, SYSDATE);  
    END log_exec;  
  
    BEGIN  
        DELETE FROM employees  
        WHERE employee_id = p_id;  
        Log_exec;  
    END leave_emp2;
```

As chamadas de execução podem acontecer tanto internamente em procedures, quanto em blocos anônimos. Veja as diferenças apresentadas no exemplo para aumentar o salário de empregados. Seguem os exemplos nos destaque:

1. No bloco anônimo:

```
DECLARE
    V_id NUMBER := 163;
BEGIN
    raise_salary(v_id);
    COMMIT;
    ...
END;
```

2. Dentro de uma *procedure*:

```
CREATE OR REPLACE PROCEDURE process_emp
IS
    CURSOR emp_cursor IS
        SELECT employee_id
        FROM employees;
    BEGIN
        FOR emp_rec IN emp_cursor
        LOOP
            Raise_salary (emp_rec.employee_id);
        END LOOP;
        COMMIT;
    END process_emp;
```

```
String ...
if(parameters.contains("name"))
    hql += " and p.name = :name"
}
if(parameters.contains("age")){
    hql += " and p.age = :age"
}
TypedQuery<Person> query = em.createQuery(hql);
query.setParameter("name", name);
query.setParameter("age", age);
List<Person> result = query.getResultList();
for(Person p : result) {
    System.out.println(p);
}
```

Figura 6 - Exceções devem ser programadas

## EXCEÇÕES

O gerenciamento das exceções em tempo de execução permite gerenciar qualquer tipo de exceção em tempo de execução, e dessa forma, permite propagar para o ambiente chamador ou tomar ações quando ocorrem.

```
RAISE_APPLICATION_ERROR(numero_erro, texto_erro)
```

- **Numero\_erro** - É o número do erro definido pelo usuário. Deve estar entre -20000 e -20999.
- **Texto\_erro** - É a mensagem definida pelo usuário.

As exceções também podem ser tratadas para problemas não contemplados pelo ORACLE. Para isso, é necessário efetuar a declaração da exceção com um nome e cláusula EXCEPTION, e ainda é necessário associá-la com um número de erro utilizando o comando: PRAGMA EXCEPTION\_INIT.

Executando uma operação no Banco de Dados, como ROLLBACK, ou customizando uma mensagem de erro, para uma exceção ORACLE pode tratar estas exceções no bloco EXCEPTION.

No exemplo que segue, o retorno da EXCEPTION ocorre ao tentar excluir um funcionário não cadastrado:

```
DELETE FROM EMP WHERE empno = v_emp_no;
IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20200, 'Func. nao existe');
END IF;
COMMIT WORK;
END exclui_funcionario;
```

Com o bloco EXCEPTION incluído no código, o mesmo exemplo apresentado ficaria da seguinte forma:

```
CREATE OR REPLACE PROCEDURE exclui_funcionario (v_emp_no IN
emp.empno%TYPE) IS
BEGIN
    DELETE FROM EMP WHERE empno = v_emp_no;
    COMMIT WORK;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20200, 'Func. nao existe');
END exclui_funcionario;
```

## EXECUTAR UMA PROCEDURE

De qualquer ambiente PL/SQL, basta simplesmente chamar a *procedure* com uma chamada direta em um bloco anônimo, como no exemplo:

```
DECLARE
    V_empno  NUMBER := 7654;
BEGIN
    ...
    Exclui_funcionario (v_empno);
END;
```

A execução de outra procedure ocorre no seguinte exemplo:

```
CREATE PROCEDURE processa_funcionario
    (v_emp_no IN emp.empno%TYPE)
IS
BEGIN
    ...
    exclui_funcionario (v_empno);
    ...
END;
```

**Observação Importante:** ao executar o PL/SQL dos ambientes SQL \*Plus ou do SQL \*DBA, é necessário utilizar o comando **EXECUTE**. E para entrar com valores via SQL \*Plus é necessário utilizar o comando ACCEPT e substituir o parâmetro de entrada da procedure pela variável do ACCEPT iniciado pela string &, conforme o exemplo:

```
ACCEPT p_empno PROMPT 'Entre com o numero do funcionario:'
EXECUTE exclui_funcionario (&p_empno);
```

De acordo com o ambiente chamador, é possível executar *procedures* de um outro usuário (*schema*) ou de um outro Banco de Dados.

No exemplo, a execução de uma *procedure* de outro usuário:

```
EXECUTE william.exclui_funcionario (7654);
```

No exemplo, a execução de uma *procedure* de outro Banco de Dados:

```
EXECUTE william.exclui_funcionario (7654)@pr;
```

No caso em que uma *procedure* contenha vários argumentos, existem três métodos para especificar seus valores:

1. **Posicional** - Lista valores na ordem em que foram declarados, exemplo:

```
EXECUTE novos_funcionarios ('WILLIAM', 'ANALISTA', 7566, 3000);
```

2. **Nomeado** - Lista valores associando cada valor com o nome do argumento utilizado na sintaxe especial, exemplo:

```
EXECUTE novos_funcionarios (v_emp_sal => 3000, v_mgr_no => 7566, v_mp_name => 'LUIZ', v_emp_job => 'ANALISTA');
```

3. **Combinado** - Lista os primeiros valores de acordo com a posição e o res-tante nomeando, exemplo:

```
EXECUTE novos_funcionarios ( 'LUIZ', 'ANALISTA', v_
emp_sal =>3000, v_mgr_no = >7566);
```

As maneiras de executar uma função são as mesmas utilizadas para executar uma *procedure*. Mas no caso de uma função o retorno é de um único valor.

Executando uma função de um bloco PL/SQL, segue exemplo:

```
DECLARE
    v_empno      NUMBER := 7654;
    v_sal        NUMBER;
BEGIN
    ...
    v_sal := pesquisa_salario (v_emp_no);
    ...
END;
```

Ao executar uma função de uma procedure, temos o seguinte código de exemplo:

```
CREATE PROCEDURE processa_emp (v_emp_no IN emp.emp-
no%TYPE) IS
    v_sal      NUMBER;
BEGIN
    ...
    v_sal := pesquisa_salario (v_emp_no);
    ...
END;
```

Uma observação importante sobre o SQL \*PLUS é que o valor retornado de uma função é armazenado em uma variável global, e a execução nessa interface segue o seguinte exemplo:

```
ACCEPT p_empno PROMPT 'Entre com o numero do funcionario: '
VARIABLE g_sal NUMBER
EXECUTE :g_sal := pesquisa_salario(&p_empno);
PRINT g_sal
```

- Para verificar o status de uma *procedure*:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS WHERE OBJECT_NAME = <nome_da_procedure>;
```

- Para visualizar o código-fonte de uma *procedure*:

```
SELECT TEXT FROM USER_SOURCE
WHERE NAME = <nome_da_procedure> ORDER BY LINE;
```

- Para eliminar uma *procedure*:

```
SHOW ERRORS PROCEDURE <nome_da_procedure>;
```

- Para eliminar uma *procedure*:

```
DROP PROCEDURE <nome_da_procedure>;
```



Figura 7 - Funções

## FUNCTIONS

Funções são blocos PL/SQL nomeados que retornam um valor, podem ser armazenadas no Banco de Dados. Uma função é chamada como parte de uma expressão.

As vantagens das funções em expressões SQL são:

1. Estender o SQL para atividades mais complexas.
2. Aumentar a eficiência para filtrar dados na cláusula WHERE.
3. Manipular dados do tipo *string*.

As funções são amplamente utilizadas e a ligação com elas são os comandos e cláusulas que auxiliam manipulações e nos resultados. Com isso, as funções podem ser utilizadas:

1. Juntamente ao comando SELECT.
2. Para valores no comando INSERT.
3. Para valores no comando UPDATE.
4. Nas cláusulas de filtro WHERE e HAVING.
5. Nas cláusulas de resultados CONNECT BY, START WITH, ORDER BY e GROUP BY.

Algumas restrições devem ser observadas para usar as funções definidas pelo usuário, pois elas devem ser armazenadas no Banco de Dados, aceitar somente parâmetros do tipo IN, aceitar somente tipos de dados válidos no SQL, não utilizando tipos de dados PL/SQL e devem retornar tipos de dados válidos no SQL.

As restrições na chamada das funções tratam de alguns detalhes importantes, com isso, as funções chamadas:

1. Com SQL não podem conter comandos DDL.
2. Por um UPDATE ou DELETE em uma tabela T não podem conter comandos DML na mesma tabela.
3. Por qualquer comando DML não podem fazer uma pesquisa na mesma tabela.
4. A partir de comandos SQL não podem conter comandos que finalizam uma transação.

A Sintaxe da *Function* é muito parecida com a sintaxe e as restrições de uma *Procedure* pois possuem a mesma construção. Apresentamos a seguir, o modelo:

```
CREATE [OR REPLACE] FUNCTION function_name
    [ (parameter1 [mode1] datatype1,
      Parameter2 [mode2] datatype2,
      . . . ) ]
    RETURN datatype
    IS | AS
    PL/SQL Block;
```

Segue também um exemplo do código da criação e da execução da *function*:

```
CREATE OR REPLACE FUNCTION GET_SAL
    (EMP_ID EMP.EMPNO%TYPE)
RETURN NUMBER
IS
    V_SAL EMP.SAL%TYPE;
BEGIN
    SELECT SAL INTO V_SAL
    FROM EMP
    WHERE EMPNO = EMP_ID;
    RETURN V_SAL;
END;
```

Após a mensagem emitida: “**Function created**”, a função pode ser executada por meio do seguinte código:

```
SELECT GET_SAL(7788) FROM DUAL;
```

O resultado é apresentado da seguinte forma:

```
GET_SAL(7788)
-----
3000
```

Na interface SQL\*PLUS, a execução da *function* é diferenciada, conforme apresentada a seguir:

```
VARIABLE SALARIO NUMBER  
EXECUTE :SALARIO := GET_SAL(7788);
```

Após a mensagem emitida: “**PL/SQL function successfully completed**”, a função pode ser executada por meio do código:

```
PRINT SALARIO
```

O resultado é apresentado da seguinte forma:

```
SALARIO
```

```
-----  
3000
```

A exclusão da *function* é simples. Seu comando possui a seguinte estrutura:

```
DROP FUNCTION <nome da function>;
```

Comparando as *Functions* e *Procedures*, podemos observar em poucos detalhes como a diferença fundamental entre ambas é o fato de funções serem caracterizadas pelo retorno de valores como parâmetros de saída.

As *functions* e *procedures* agregam os benefícios da fácil manutenção, a melhora na segurança e integridade dos dados, a melhora na performance e também no código.

A comparação entre *Procedure* e *Function* aborda detalhes de como é a aplicação de cada uma e também a sua forma de retornar informação. Esses detalhes podem ser observados no quadro a seguir.

Quadro 2 - Comparação de aplicabilidade entre *Procedure* e *Function*

PROCEDURE	FUNCTION
Executa como um comando PL/SQL.	É chamada como parte de uma expressão.
Não existe a cláusula RETURN no cabeçalho.	Precisa conter o cláusula RETURN.
Pode retornar nenhum um ou mais valores.	Retorna apenas um valor.
	É necessário que o último comando seja o RETURN.

Fonte: os autores.

- Para verificar o status de uma função:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS WHERE OBJECT_NAME = <nome_da_funcao>;
```

- Para visualizar o código-fonte de uma função:

```
SELECT TEXT FROM USER_SOURCE
WHERE NAME = <nome_da_funcao> ORDER BY LINE;
```

- Para eliminar uma função:

```
SHOW ERRORS FUNCTION <nome_da_funcao>;
```

- Outra forma de eliminar uma função:

```
DROP FUNCTION <nome_da_funcao>;
```

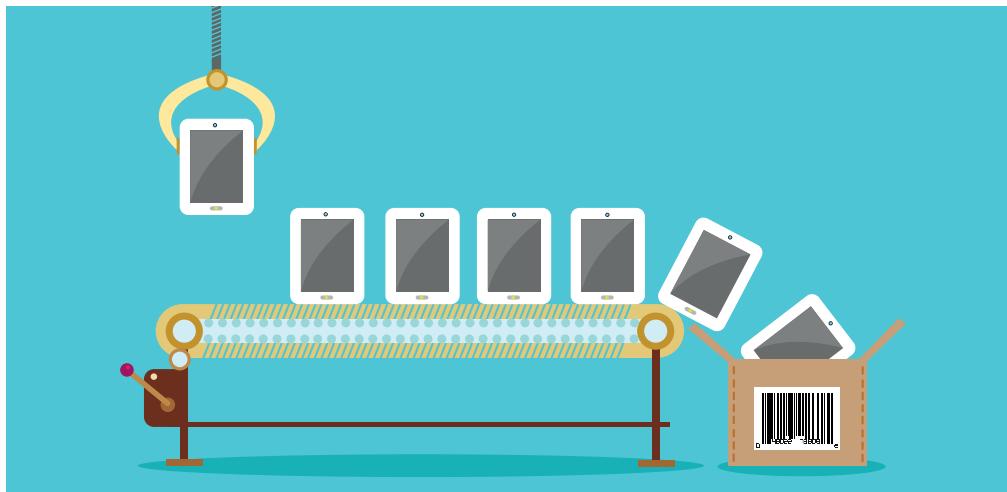


Figura 8 - Ilustração de pacote de tipos, itens e subprogramas

## PACKAGES

*Packages* são grupos lógicos de tipos de PL/SQL, itens e subprogramas. Possuem duas partes: Especificação (*specification*) e Corpo (*body*) e não podem ser chamados diretamente ou parametrizados. Entretanto, é importante destacar que permitem ao servidor colocar ou ler múltiplos objetos na memória de uma única vez.

A especificação do *package* possui variáveis públicas, *procedures* públicas e a declaração da *procedure*.

Os componentes do corpo do *package* são compostos pela definição de *procedures*, por *procedures* públicas ou privadas e também por variáveis locais ou privadas.

Os passos básicos para desenvolver um *package* são similares aos de desenvolver uma *procedure*.

É recomendado salvar os textos de especificação e corpo em arquivos diferentes para facilitar alterações posteriores. O desenvolvimento do *package* utiliza comandos como “CREATE PACKAGE”, “CREATE PACKAGE BODY”, além de poder chamar construções públicas dentro de pacotes em ambiente Oracle.

A sintaxe utilizada na criação da especificação e do corpo do *package* é apresentada nos códigos seguintes, e o REPLACE é uma opção para incluir quando o pacote já existir. Segue a sintaxe:

```
CREATE [ OR REPLACE ] PACKAGE PACKAGE_NAME
IS | AS
    DECLARAÇÃO DE ITENS E TIPOS PÚBLICOS
    ESPECIFICAÇÕES DE SUBPROGRAMAS
END PACKAGE_NAME;

CREATE [OR REPLACE] PACKAGE BODY PACKAGE_NAME
IS | AS
    PRIVATE TYPE AND ITEM DECLARATIONS
    SUBPROGRAM BODIES
END PACKAGE_NAME;
```

Da mesma forma, para exemplificar a criação do *package*, serão apresentados os códigos de um exemplo que tratam de comissão. Estão apresentados os códigos da especificação e do corpo. Também é importante observar que é possível chamar uma função ou *procedure* no mesmo *package*, assim como na *procedure* criada RESET\_COMM, que chama a *function* VALIDATE\_COMM, que foi criada no mesmo PACKAGE, segue o exemplo:

```
CREATE OR REPLACE PACKAGE comm_package
IS
    G_COMM NUMBER := 0.10; -- inicializando com 0,10
    PROCEDURE RESET_COMM (P_COMM IN NUMBER);
END comm_package;
---> Corpo do Package
CREATE OR REPLACE PACKAGE BODY comm_package
IS
    FUNCTION VALIDATE_COMM(P_COMM IN NUMBER)
    RETURN BOOLEAN
    IS
        V_COMM NUMBER;
    BEGIN
        SELECT MAX(COMM) INTO V_COMM FROM
EMP;
        IF P_COMM > V_COMM THEN
            RETURN(FALSE);
        ELSE
            RETURN(TRUE);
        END IF;
    END VALIDATE_COMM;
    PROCEDURE RESET_COMM (P_COMM IN NUMBER)
    IS
    BEGIN
        IF VALIDATE_COMM(P_COMM) THEN
            G_COMM := P_COMM;
        ELSE
            RAISE_APPLICATION_ERROR (-20210,
'Comissão inválida.');
        END IF;
    END RESET_COMM;
END comm_package;
```



Figura 9 - Imagem da marca Oracle

Quadro 3 - Alguns *packages* oferecidos pela Oracle

PACKAGE	DESCRÍÇÃO
DBMS_OUTPUT	Saída de informação de procedure ou funções armazenadas.
DBMS_DDL	Compila <i>procedure</i> , funções e <i>packages</i> . Obtém estatísticas de performance com o comando <i>analyse</i> .
DBMS_SESSION	Altera a sessão do usuário. Define regras para o usuário e reinicializa o estado de um <i>package</i> .
DBMS_PIPE	Envia mensagem do Banco de Dados para a aplicação.
DBMS_JOB	Gerencia o agendamento de tarefas no Banco de Dados.
UTL_FILE	Trabalha com arquivos textos.
UTL_TCP	Habilita o Oracle a se comunicar com outros servidores.

Fonte: os autores.

Um package também pode não ter o BODY e pode ser executado normalmente, veja no exemplo a seguir:

```
CREATE OR REPLACE PACKAGE global_consts IS
    mile_2_kilo CONSTANT NUMBER := 1.6093;
    kilo_2_mile CONSTANT NUMBER := 0.6214;
    yard_2_meter CONSTANT NUMBER := 0.9144;
    meter_2_yard CONSTANT NUMBER := 1.0936;
END global_consts;

EXECUTE DBMS_OUTPUT.PUT_LINE( '20 miles = ' || 20 *
    Global_consts.mile_2_kilo || 'km' )
```

Em um *package* também pode haver sobrecarga (*overloading*), que habilita a utilização de um mesmo nome para várias *procedures* ou *functions* dentro do mesmo *package*. Nesse caso, é necessário ter parâmetros formais que diferenciem o método, mas somente é possível a sobrecarga de métodos dentro do package, por exemplo:

```
CREATE OR REPLACE PACKAGE over_pack
IS
    PROCEDURE add_dept
        (p_deptno IN departments.department_id%TYPE,
         p_name   IN departments.department_name%TYPE
                     DEFAULT 'unknown',
         p_loc    IN departments.location_id%TYPE DEFAULT 0);
    PROCEDURE add_dept
        (p_name   IN departments.department_name%TYPE
                     DEFAULT 'unknown',
         p_loc    IN departments.location_id%TYPE DEFAULT 0);
END over_pack;
```

```
CREATE OR REPLACE PACKAGE BODY over_pack IS
    PROCEDURE add_dept
        (p_deptno IN departments.department_id%TYPE,
         p_name   IN departments.department_name%TYPE
                    DEFAULT 'unknown',
         p_loc    IN departments.location_id%TYPE DEFAULT 0)
    IS
    BEGIN
        INSERT INTO departments (department_id, department_name,
                               location_id)
        VALUES  (p_deptno, p_name, p_loc);
    END add_dept;
    PROCEDURE add_dept
        (p_name   IN departments.department_name%TYPE
                    DEFAULT 'unknown',
         p_loc    IN departments.location_id%TYPE DEFAULT 0)
    IS
    BEGIN
        INSERT INTO DEPARTMENTS(department_id,
                               department_name, location_id)
        VALUES (departments_seq.NESTVAL, p_name, p_loc);
    END add_dept;
END over_pack;
```

Para excluir um *package* é necessário excluir a especificação e o corpo. Para isso é utilizado o comando **DROP PACKAGE <nome>** para a especificação e **DROP PACKAGE BODY <nome>** para o corpo do *package*.

## CONSIDERAÇÕES FINAIS

Prezado(a) aluno(a)! Nesta unidade, foi possível reunir conceitos relacionados a Banco de Dados de nível mais avançado, com a apresentação de códigos e exemplos necessários para a exploração do assunto.

O nosso estudo promoveu a compreensão de que subprogramas são blocos PL/SQL com nomes e declarados como *procedure* ou **functions**, podendo ser blocos anônimos e o subprograma pode ser chamado de diferentes ambientes.

Dentre os benefícios dos subprogramas, constam o fácil gerenciamento, a melhora na segurança e na integridade dos dados, a melhora na performance do sistema e clareza no código.

Também as *procedures*, que são subprogramas e executam ações programadas, podem ser criadas, compiladas e armazenadas no Banco de Dados. Parâmetros são utilizados para enviar dados do ambiente chamador para a procedure de três formas: IN, OUT e IN OUT.

Um subprograma pode ser declarado dentro de outro, contudo *procedures* somente podem ser chamadas pelos ambientes que possuem suporte, e uma *procedure* pode ser facilmente excluída com o comando DROP PROCEDURE.

Uma função, igualmente à *procedure*, é um bloco PL/SQL que retorna um valor. É criada por meio do comando CREATE FUNCTION e pode ser chamada por um comando SQL, como parte de uma expressão.

Para a função ser usada como comandos SQL, deverá estar armazenada no BD, e pode ser removida facilmente por meio do comando DROP FUNCTION.

É comum a utilização de procedures para executar uma ação e de *functions* para cálculos de valores.

Já uma *package* melhora a segurança, organização, gerenciamento e performance. Nela podem estar juntos grupo de *procedures* e funções e assim garantindo a segurança de acesso ao *package*.

Encerrando, temos as *triggers*. Como efetuar a sua criação, os eventos que podem ser associados ao *trigger*, seu gerenciamento e a possibilidade de utilização em auditorias.

Mais adiante, serão detalhadas as questões de controle de acesso ao Banco de Dados e aos códigos de programação SQL, de forma a oferecer conhecimentos úteis para o dia a dia na administração desses processos de Bancos de Dados.

## ATIVIDADES



1. Dada a tabela valor\_produto e a trigger verifica\_valor, abaixo:

```

CREATE TABLE valor_produto
    (codigo NUMBER(4),
     valor_anterior NUMBER(7,2),
     valor_novo NUMBER(7,2) );

CREATE OR REPLACE TRIGGER verifica_valor
BEFORE UPDATE
OF valor
ON produto
FOR EACH ROW
BEGIN
    INSERT INTO valor_produto
    VALUES (:OLD.codigo, :OLD.valor, :NEW.valor);
END

```

CODIGO	VALOR_ANTERIOR	VALOR_NOVO
3	5,8	5,4

**Incluir na tabela valor\_produto os campos:**

**codigo\_usuario VARCHAR2 (30)**

**data\_log DATE**

Alterar a *trigger verifica\_valor* para que também sejam incluídos na tabela valor\_produto, a data do sistema no momento da atualização e o nome do usuário que realizou a alteração no campo valor.

## ATIVIDADES



2. Criar uma função para apresentar o fatorial de um número a ser informado no comando SELECT. Lembrete:  $x! = x * (x-1)!$

No exemplo:

```
SELECT factorial (3) FROM dual;
```

O retorno apresentado deverá ser:

FATORIAL (3)

6

3. Crie uma tabela chamada CIRCULO com as seguintes colunas:

```
RAIO NUMBER (2) ,  
AREA NUMBER (8, 2)
```

```
CREATE TABLE CIRCULO (  
    RAIO NUMBER(2) ,  
    AREA NUMBER(8, 2) );
```

Desenvolva um programa em PL/SQL para inserir os raios com valores 1 a 10 e as respectivas áreas na tabela criada utilizando WHILE ou FOR.

4. Conforme tabela de erros abaixo, segue:

ERRO	NOME	DESCRÍÇÃO
ORA - 00001	DUP_VAL_ON_INDEX	Tentativa de armazenar valor duplicado em uma coluna que possui chave primária ou única.
ORA - 01403	NOT_LOGGED_ON	Tentativa acessar o banco de dados sem estar conectado a ele.
ORA - 01403	NO_DATA_FOUND	Ocorre quando um comando SELECT ... INTO não retorna nenhuma linha.
ORA - 01422	TOO_MANY_ROWS	Ocorre quando um comando SELECT ... INTO retorna mais de uma linha.
ORA - 01476	ZERO_DIVIDE	Tentativa de dividir qualquer número por zero.

Fonte: os autores.

## ATIVIDADES



Elabore um programa em PL/SQL que faça o tratamento de exceção na tabela ALUNO, cujo o código de criação é:

```
CREATE TABLE ALUNO (
    RA NUMBER(9),
    NOME VARCHAR2(30));
INSERT INTO ALUNO VALUES (1,'MARIA');
```

No programa em PL/SQL, Informe a tentativa de inserir valor duplicado numa coluna que é chave primária.

5. Utilizando as tabelas de exemplo citadas nesta unidade, faça uma *procedure* para cadastrar um departamento. O departamento deve ser incrementado por meio de um select no maior dept + 10.



O DataBase Administrator (DBA), é o Administrador de Banco de Dados e o gerente responsável pelos sistemas de Banco de Dados. É o responsável pelos privilégios de acesso e de classificação de usuários do sistema conforme as determinações das políticas de segurança.

O DBA possui uma conta direitos de acesso próprio e acessa a toda e qualquer instância do Banco de Dados. Sua função é manter o Banco íntegro, tanto em relação às contas de acesso quanto à consistência dos dados e ao andamento da performance do banco.

Outra função importante do DBA é a de se encarregar do tuning do Banco de Dados, referente à performance de operação do banco. Responsável também pela cópia de segurança dos dados e pela criação de índices, *triggers*, *stored procedures* e demais estruturas necessárias.

Toda esta estrutura e cessão de direitos demonstra a grandeza da segurança do sistema de gerenciamento de Banco de Dados, em especial o Oracle.

Os tipos de segurança devem levar em consideração os perfis de segurança de acesso aos dados que devem ser traçados. Para proteger o BD, e atingir as medidas de segurança necessárias, ações devem ser tomadas em diversos níveis.

- Físico – torna o sistema fisicamente seguro contra entradas de intrusos.
- Humano – os controles dos acessos dos usuários são cuidadosamente estudados.
- Sistema operacional – a fragilidade na segurança do SO pode ser uma porta de acesso não-autorizado ao Banco de Dados.
- Sistema de BD – os usuários de sistemas de BD devem ter autorização de acesso somente nas porções limitadas e necessárias. Outros usuários deverão ser habilitados a emitir consultas, com a proibição de modificar dados.

A metodologia e *tuning* da Oracle, é focada no design da aplicação e no *tuning* de consultas SQL mesmo antes de analisar qualquer tipo de problema relacionado à configuração do Banco de Dados ou à ação do DBA.

A otimização de uma consulta determina a melhor estratégia ou forma de execução por parte do Banco de Dados. O otimizador do Oracle é eficiente na escolha, e faz, por exemplo, uma análise se usará um índice ou não para uma determinada consulta e quais as técnicas de JOIN usar na junção de tabelas. Estas decisões impactam diretamente na performance de um SQL.

É muito importante que os conceitos do otimizador do Oracle e de tuning sejam conhecidos pelos desenvolvedores. Tal conhecimento ajudará a escrever consultas mais eficientes, rápidas e que não impactarão nas atividades do DBA.

Fonte: os autores.

# MATERIAL COMPLEMENTAR



NA WEB

A Oracle disponibiliza um artigo em inglês com as principais diferenças entre o Oracle e o MySQL, intitulado "Database SQL Developer Supplementary Information for MySQL Migrations". Nele estão as descrições dos objetos, migrações e a comparação de comandos.

Acesse o link a seguir:

<[http://docs.oracle.com/cd/E12151\\_01/doc.150/e12155/oracle\\_mysql\\_compared.htm#CHDIIBJH](http://docs.oracle.com/cd/E12151_01/doc.150/e12155/oracle_mysql_compared.htm#CHDIIBJH)>



# REFERÊNCIAS

## Referências on-line

- 1- Em: <<http://docslide.com.br/documents/apostila-introducao-ao-oracle -procedural-option-para-amigos.html>>. Acesso em: 25 nov. 2016.



# GABARITO

1)

```
ALTER TABLE valor_produto ADD (codigo_usuario VARCHAR2(30), data_
log DATE);

CREATE OR REPLACE TRIGGER verifica_valor
BEFORE UPDATE
OF valor
ON produto
FOR EACH ROW
BEGIN
    INSERT INTO valor_produto
    VALUES (:OLD.codigo, :OLD.valor, :NEW.valor, user, sysdate);
END;
/
```

2)

```
CREATE OR REPLACE FUNCTION fatorial (p_n IN NUMBER)
RETURN number
IS
BEGIN
    IF p_n = 1 THEN
        RETURN 1;
    ELSE
        RETURN p_n * fatorial (p_n-1);
    END IF;
END fatorial;
```

3)



# GABARITO

## SOLUÇÃO 1: WHILE

```

DECLARE
    PI CONSTANT NUMBER(9,7) := 3.1415927;
    RAIO NUMBER(2);
    AREA NUMBER(8,2);
BEGIN
    RAIO := 1;
    WHILE RAIO <=10
    LOOP
        AREA := PI*POWER(RAIO,2);
        INSERT INTO CIRCULO VALUES (RAIO,AREA);
        RAIO := RAIO+1;
    END LOOP;
END;

```

OU

## SOLUÇÃO 2: FOR

```

DECLARE
    PI CONSTANT NUMBER(9,7) := 3.1415927;
    RAIO NUMBER(2) := 1;
    AREA NUMBER(8,2);
BEGIN
    FOR CONTADOR IN 1..10
    LOOP
        AREA := PI*POWER(RAIO,2);
        INSERT INTO CIRCULO VALUES (RAIO,AREA);
        RAIO := RAIO +1;
    END LOOP;
END;

```



# GABARITO

4)

```
DECLARE
BEGIN
    INSERT INTO ALUNO VALUES (1,'ANTONIO');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE ('Já existe um aluno com este RA');
END;
```

5)

```
CREATE OR REPLACE PROCEDURE INS_DEPT
    (P_DNAME DEPT.DNAME%TYPE, P_LOC DEPT.LOC%TYPE)
IS
    V_DEPT_EMP EMP.DEPTNO%TYPE;
BEGIN
    SELECT MAX(DEPTNO) INTO V_DEPT_EMP FROM DEPT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN V_DEPT_EMP := 0;
END;
V_DEPT_EMP := V_DEPT_EMP + 10;
INSERT INTO DEPT VALUES (V_DEPT_EMP, P_DNAME, P_LOC);
END;
```



# CONTROLANDO ACESSOS

UNIDADE

V

## Objetivos de Aprendizagem

- Gerenciar usuários.
- Controlar os papéis para gerenciar acesso aos dados.
- Usar comando DCL para controlar acesso a privilégios de sistema e de objetos.
- Utilizar instruções para gerenciar direitos de acesso a Banco de Dados.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- *Triggers* (gatilhos)
- Segurança do Banco
- Controle de acesso ao usuário
- Criar e acessar vínculos de Banco de Dados
- Gerenciando Senhas e Recursos
- Gerenciando Usuários
- Auditoria



## INTRODUÇÃO

Olá, caro(a) aluno(a)! Com o valor agregado aos dados pelas empresas, é possível mensurar o grau de importância que os bancos de dados recebem para que a proteção devida seja atribuída a eles. Os Bancos de Dados precisam ser especialmente protegidos já que guardam tantas possibilidades e detalhes da empresa.

Em relação ao armazenamento, tratar e conservar informações, é um dos assuntos essenciais em Banco de Dados. Por isso, o tema abordado toma grandes proporções, pois é possível imaginar que essa seja uma questão difícil de assimilar.

Entretanto, o que deve ser levado em consideração é que os dados ou informações de uma empresa, podem ser tão importantes para o andamento dos processos cotidianos que impactam diretamente no sucesso ou no seu fracasso. Dessa forma, a segurança se torna uma questão tão importante quanto os próprios dados do sistema.

Ou seja, desde a criação da estrutura para receber os dados, até os perfis e os níveis de acesso, tudo deve ser devidamente pensado e estruturado para que a garantia de uso e segurança de acesso sejam considerados. O Administrador de Banco de Dados (DBA) é o profissional responsável pelo gerenciamento de um sistema de Banco de Dados. Responsável também pela concessão de privilégios, é ele quem libera os acessos e classifica os usuários do sistema, conforme as determinações das políticas de segurança.

O Banco de Dados Oracle segue em grande evolução, e cada vez mais se torna autogerenciável por meio de novos recursos adicionados que o tornam capaz de realizar certas tarefas sem a necessidade da interação humana. Tudo deve ser planejado dentro da política de segurança da empresa, e programado e concordância com o DBA, Gerência e até a Governança de TI, conforme o tamanho da empresa e sua divisão setorial.

Nesta unidade serão abordados os diversos pontos que tratam da segurança no Banco de Dados. Esses pontos tratam diretamente da administração e da proteção dos dados, que em primeira instância contam com o gerenciamento de um DBA para a garantia de segurança e disponibilidade.

Bom estudo!

## TRIGGERS (GATILHOS)

Um *trigger* ou gatilho é um bloco *PL/SQL* (Procedural Language/Structured Query Language) associado a uma tabela. Executa implicitamente de acordo com um evento e podem ser de aplicação ou de Banco de Dados. Desenvolver um *gatilho* de Banco de Dados é solicitar a execução de um bloco *PL/SQL* somente quando um comando de manipulação específico é executado em uma certa tabela. Algumas diferenças elementares entre gatilho e *procedure* (procedimento) são expressadas na tabela 1.

Tabela 1 - Diferenças entre *Triggers* e *Procedures*

TRIGGERS	PROCEDURE
Definido com <i>CREATE TRIGGER</i> .	Definido com <i>CREATE PROCEDURE</i> .
Código na view <i>USER_TRIGGER</i> .	Código na view <i>USER_SOURCE</i> .
Chamada automática conforme o evento.	Necessário fazer a chamada.
<i>COMMIT, SAVEPOINT</i> e <i>ROLLBACK</i> não são permitidos.	<i>COMMIT, SAVEPOINT</i> e <i>ROLLBACK</i> são permitidos.

Fonte: os autores.

O uso de *trigger* é recomendado para desenvolver regras complexas de validação de dados que não podem ser criadas com as *constraints*. Também é possível fazer replicação de dados síncrona e ainda é utilizada para fazer auditorias.

A decisão de qual o momento em que o evento do *trigger* deve ser executado deve ser definido para que ocorra antes ou depois do “disparo” do evento. As opções estão apresentados na tabela 2.

Tabela 2 - Tabela de partes e valores de um *Trigger*

PARTES	DESCRIÇÃO	VALORES POSSÍVEIS
Momento de Execução	Quando o <i>trigger</i> é disparada em relação ao evento	Tabelas: <i>BEFORE</i> ou <i>AFTER</i> Views: <i>INSTEAD OF</i>
Evento	Qual operação de manipulação dos dados na tabela causará o disparo do trigger	<i>INSERT</i> <i>UPDATE</i> <i>DELETE</i>
Tipo	Quantas vezes o corpo do trigger será executado	Para cada registro ou para o comando
Cláusula	Causa uma restrição para a execução do <i>trigger</i>	<i>WHEN</i>
Corpo	Quais ações que o trigger executará	Bloco de código <i>PL/SQL</i>

Fonte: os autores.

A sintaxe do *trigger* é mostrada a seguir para ilustrar como são utilizados na codificação de um sistema de Banco de Dados.

```
CREATE [OR REPLACE] TRIGGER trigger_gatilho
    timing
        evento1 [OR evento2 OR evento3]
        ON tabela
    gatilho_corpo
```

Um exemplo interessante é a proibição de incluir um funcionário após o horário comercial mostrado a seguir, em que é verificado o horário de tentativa de acesso de um funcionário de acordo com os horários permitidos.

```
CREATE OR REPLACE TRIGGER SECURE_EMP
BEFORE INSERT ON EMP
BEGIN
  IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
    (TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) NOT BETWEEN 8 AND
     18) THEN
    RAISE_APPLICATION_ERROR(-20500, 'Não é possível incluir
      um funcionário fora do horário comercial');
  END IF;
END;
```

Outro exemplo da combinação de eventos dentro de um *trigger* é mostrado a seguir, em que são acrescentadas ações de acordo com a sequência de eventos gerados pelos gatilhos.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE
OR DELETE ON emp
BEGIN
  IF
    (TO_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN'))
  OR
    (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '18')
  THEN
    IF DELETING THEN
      RAISE_APPLICATION_ERROR (-20502, 'Deletar no
        horário comercial.')
    ELSE IF INSERTING THEN
      RAISE_APPLICATION_ERROR (-20500, 'Inserir no
        horário comercial.')
    END IF;
  END IF;
END;
```

```
ELSE IF  UPDATING ('SALARY')  THEN
    RAISE_APPLICATION_ERROR (-20503,'Atualizar no
horário cial.')
ELSE
    RAISE_APPLICATION_ERROR (-20504,'Atualizar no
horário cial.')
END IF;
END IF;
END;
```

Criando um *trigger* para cada registro a ser inserido ou que tenha seus dados atualizados, conforme o código:

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON  emp FOR EACH ROW
BEGIN
    IF NOT (:NEW.job IN ('AD_PRES', 'AD_VP'))
        AND: NEW.sal < 15000
    THEN
        RAISE_APPLICATION_ERROR (-20202, 'Valor não
autorizado.')
    END IF;
END;
```

Usando os qualificadores *OLD* e *NEW* o código seguinte registra uma auditoria na manutenção de registros, gravando os velhos e os novos valores conforme indicado no exemplo:

```
CREATE OR REPLACE TRIGGER audit_emp
AFTER DELETE OR INSERT OR UPDATE ON emp FOR EACH ROW
BEGIN
    INSERT INTO AUDIT_EMP (USER_NAME, DATA, EMPNO,
    OLD_ENAME, NEW_ENAME,
    OLD_JOB, NEW_JOB,
    OLD_SAL, NEW_SAL)
    VALUES (USER, SYSDATE, :OLD.EMPNO,
            :OLD.ENAME, :NEW.ENAME,
            :OLD.JOB, :NEW.JOB,
            :OLD.SAL, :NEW.SAL);
END;
```

Um exemplo de código para restringir a ação do *trigger* é mostrado da seguinte maneira:

```
CREATE OR REPLACE TRIGGER derive_comm_pct
BEFORE INSERT OR UPDATE OF SAL ON emp FOR EACH ROW WHEN (NEW.
JOB = 'SALESMAN')
BEGIN
    IF INSERTING THEN
        :NEW.COMM := 0;
    ELSEIF :OLD.COMM IS NULL THEN
        :NEW.COMM := 0;
    ELSE
        :NEW.COMM := :OLD.COMM + 0.05;
    END IF;
END;
```

Um gatilho pode ter o seu funcionamento habilitado ou desabilitado. Isso é muito útil quando for necessário executar algumas instruções que não necessitem que os *triggers* sejam executados. Dessa forma, os comandos são:

```
ALTER TRIGGER <nome> DISABLE/ENABLE  
ALTER TABLE <nome> DISABLE/ENABLE ALL TRIGGERS;  
ALTER TRIGGER <nome> COMPILE;
```

Para remover um *trigger* é só utilizar o seguinte comando mostrado:

```
DROP TRIGGER <nome>;
```

Contudo, é necessário destacar que ao remover uma tabela, seus gatilhos serão automaticamente excluídos.

## SEGURANÇA DO BANCO

O responsável pelo gerenciamento de um sistema de Banco de Dados é o Administrador de Banco de Dados (DBA), que é a pessoa capaz de conceder privilégios de acesso e classificação de usuários do sistema conforme as determinações das políticas estabelecidas para segurança.

O DBA possui direitos de acesso a toda e qualquer instância do Banco de Dados com sua conta própria. O seu papel é garantir a integridade do Banco, seja em relação às contas de acesso e também à consistência dos dados, além da *performance* do Banco, encarregando-se do *tunning* do Banco. Também administra as cópias de segurança dos dados, a criação de índices, *triggers*, *stored procedures* e outros.



## REFLITA

Garantir a segurança da informação é fazer com que as informações permaneçam confidenciais, íntegras e disponíveis. Os Sistemas de Gerenciamento de Banco de Dados podem garantir os princípios da segurança por meio das nuvens?

Quando é necessário o acesso de um novo usuário, ou a modificação de perfis de acesso de existentes, o DBA cria a nova conta, atribui as permissões necessárias ou modifica as configurações de contas existentes mantendo assim, o acesso e controle das ações relativas a segurança de acesso concentradas em uma única pessoa com perfil adequado.

## CONTROLE DE ACESSO AO USUÁRIO

O DBA determina o controle do acesso do usuário, seus privilégios como usuário em relação ao sistema e áreas com permissão de acesso e trabalho, mantendo sob controle as ações dos colaboradores.

Esses privilégios tratam da segurança do Banco de Dados, seja em sistema ou em dados, bem como o acesso ao Banco de Dados, manipulação de conteúdos no Banco de Dados e ainda manipulação de *schemas* que tratam das coleções de objetos, como tabelas, visões e sequências.

Com cerca de 100 tipos de configurações de acesso e manipulação do Banco de Dados, o DBA tem o mais alto nível de acesso ao sistema, sendo capaz de ações como criar novos usuários, remover usuários, remover qualquer tabela, e fazer backup das tabelas (ORACLE, 2004, on-line).<sup>1</sup>

## CRIANDO USUÁRIOS

O DBA cria usuários utilizando a instrução *CREATE USER* que permite a identificação de um usuário e sua senha de acesso ao sistema como mostramos a seguir:

Sintaxe:

```
CREATE USER usuário  
IDENTIFIED BY senha;
```

Exemplo:

```
CREATE USER funcionarioscott  
IDENTIFIED BY senhapadraotiger;  
User created.
```

## CONCEDENDO PRIVILÉGIOS

Quando o usuário é criado, o DBA pode conceder a ele privilégios de sistemas específicos, por exemplo, um desenvolvedor de aplicações pode ter privilégios de sistema para criar sessões, tabelas, gatilhos, procedimentos usando comandos como *CREATE SESSION*, *CREATE TABLE*, *CREATE SEQUENCE*, *CREATE VIEW*, *CREATE PROCEDURE*, e *CREATE TRIGGER*.

Ao conhecer todos os privilégios que um sistema possui, o DBA pode conceder a um usuário os privilégios individuais de sistemas específicos, usando comandos como indicado a seguir.

```
GRANT create session, create table,  
create sequence, create view  
TO scott;
```

## ROLE (FUNÇÃO)

A alocação de privilégios pode ocorrer de duas formas, sendo a primeira sem atribuição direta para cada usuário, e a segunda, com privilégios em uma atribuição, podendo agrupar usuários, assim como a alocação de privilégios para todos os gerentes.

*Role* é um papel criado pelo DBA para facilitar o gerenciamento de direitos. Torna mais fácil garantir e revogar direitos para um grupo de usuários que possuam as mesmas regras (*Role*).

O DBA cria uma *role* com o comando *CREATE ROLE*.

Para criar uma *role* e dar direitos, devem ser executados os seguintes passos:

É possível criar uma atribuição usando o comando “*CREATE ROLE MANAGER;*” e conceder privilégios a uma atribuição com o comando “*GRANT CONNECT, RESOURCE TO MANAGER;*”, além de poder conceder uma atribuição aos usuários por meio do comando “*GRANT MANAGER TO CURSO;*”.

Um usuário pode ter várias *roles* associadas. O padrão é que todas as *roles* associadas sejam habilitadas já no *logon*, sem a necessidade de senha. Aos usuários podem ser aplicadas uma ou mais *roles* padrões com o comando *ALTER USER*.

As *roles* padrões somente podem ser especificadas após a operação de *grant*. As demais *roles* deverão ser habilitadas na sessão do usuário.

Uma *role* pode ser habilitada ou desabilitada por meio da *procedure DBMS\_SESSION.SET\_ROLE*.

As *roles* padrões já são habilitadas no *login* do usuário e uma senha poderá ser necessária para habilitar uma *role* para o usuário. Para aplicar uma *role*, podemos ter instruções como “*SET ROLE GERENTE IDENTIFIED BY OOX;*”, “*SET ROLE ALL EXCEPT GERENTE;*”, e “*SET ROLE VENDEDOR;*”.

Ao remover uma *role* deve ser observada se a opção é do usuário com o comando *REVOKE*, ou do sistema com o comando *DROP*. Nesse caso, podemos utilizar instruções como “*REVOKE GERENTE FROM BOB;*”, e “*DROP ROLE GERENTE*.”

As chamadas visões do *data dictionar* podem ser:

- *DBA\_ROLES*
- *DBA\_ROLE\_PRIVS*
- *ROLE\_ROLE\_PRIVS*

- DBA\_SYS\_PRIVS
- ROLE\_SYS\_PRIVS
- ROLE\_TAB\_PRIVS
- SESSION\_ROLES

## ALTERAÇÃO DE SENHA

Para alterar a senha do usuário, o DBA cria uma conta e a inicia com uma senha padrão. A senha pode ser alterada com o comando *ALTER USER*. Por exemplo, podemos utilizar comando “*ALTER USER* pessoa IDENTIFIED BY idade;”.

## PRIVILÉGIOS DE OBJETO

Os privilégios podem ser atribuídos conforme a necessidade e suas possibilidades de acesso, podendo variar de um objeto para outro. O proprietário de um objeto detém todos os privilégios sobre ele e pode repassar privilégios específicos de seus objetos para outros usuários. Na tabela 3, está a relação dos objetos e seus respectivos privilégios possíveis.

Tabela 3: Tabela privilégios de objetos

Privilégio	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√	√		
SELECT	√	√	√	
UPDATE	√	√		

Fonte: os autores.

O usuário que recebe o direito pode repassar para os demais, mas o direito garantido é revogado caso o concedor do direito com *WITH GRANT OPTION* retire a concessão.

Não pode ser usado para um *role* a opção *WITH GRANT OPTION*, mas para simbolizar o acesso público, pode ser usado *PUBLIC*. A sintaxe de privilégios de objetos pode ser observada a seguir.

```
GRANT      priv_objeto  [(columns)]
ON         object
TO         {user | role | PUBLIC}
[WITH GRANT OPTION];
```

O exemplo de concessão de privilégios de consulta na tabela *EMPLOYEES* pode ser observado no código mostrado a seguir.

```
GRANT      select
ON         uncionarios
TO         pessoa, gerente;
```

A concessão aos usuários e atribuições de privilégios para atualização de colunas específicas pode ser observado no código a seguir:

```
GRANT      update (department, id)
ON         departamento
TO         pessoa, gerente;
```

Uma forma de garantir privilégios para outros usuários é apresentado no código que segue:

```
GRANT SELECT  
    ON emp  
    TO CURSO;
```

Apresentada a conexão do usuário e demonstração da tabela liberada do usuário PESSOA. Também é apresentado na sequência a forma de criar um *synonym* e reduzir o nome da tabela agregada ao nome do usuário, “**CONN CURSO/ALUNO**”.

```
SELECT * FROM SCOTT.EMP;  
CREATE SYNONYM EMP FOR SCOTT.EMP;  
SELECT * FROM EMP;
```

A confirmação de privilégios concedidos pode ser consultada por meio de *views* do dicionário de dados. Essas *views* podem ser conhecidas como:

- ROLE\_SYS\_PRIVS - Privilégios de sistema concedidos a atribuições.
- ROLE\_TAB\_PRIVS - Privilégios de tabela concedidos a atribuições.
- USER\_ROLE\_PRIVS - Atribuições acessíveis ao usuário.
- USER\_TAB\_PRIVS\_MADE - Privilégios de objeto concedidos sobre os objetos do usuário.
- USER\_TAB\_PRIVS\_REC - Privilégios de objeto concedidos ao usuário.
- USER\_COL\_PRIVS\_MADE - Privilégios de objeto concedidos sobre as colunas dos objetos do usuário.
- USER\_COL\_PRIVS\_REC - Privilégios de objeto concedidos ao usuário sobre colunas específicas.
- USER\_SYS\_PRIVS - Lista os privilégios de sistema concedidos ao usuário.

## RETIRAR DIREITOS

O direito de um privilégio dado à um usuário pode ser retirado. Utilizar a instrução *REVOKE* para revogar os privilégios concedidos a outros usuários. Os privilégios concedidos a outros usuários por meio da cláusula *WITH GRANT OPTION* também serão revogados. A sintaxe dessa instrução é:

Um exemplo de uso da instrução é mostrado a seguir.

```
REVOKE {privilege [, privilege...]|ALL}
      ON          object
      FROM        {user [, user...]|role|PUBLIC}
      [CASCADE  CONSTRAINTS];
```

Visões do data dictionary pode ser representadas por DBA\_SYS\_PRIVS, SESSION\_PRIVS, DBA\_TAB\_PRIVS, e DBA\_COL\_PRIVS.

## CRIAR E ACESSAR VÍNCULOS DE BANCO DE DADOS

Uma conexão de vínculo de Banco de Dados permite que os usuários locais accessem dados em um Banco de Dados remoto. Um exemplo para acesso a um banco de dados remoto, por meio de vínculo por um Banco de Dados local pode ser observado na Figura 1:

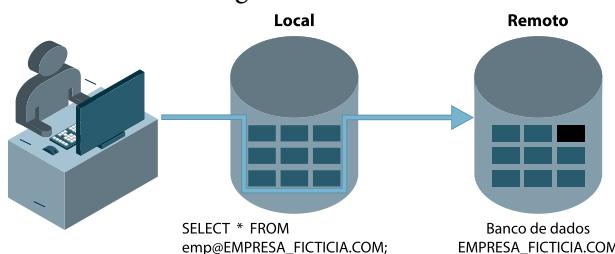


Figura 1 - Representação de Banco de dados remoto.

Fonte: os autores.

Para criar um vínculo de Banco de Dados, pode-se utilizar como exemplo, o código ”CREATE PUBLIC DATABASE LINK empresa\_exemplo.com USING ‘vendas’;”.

Para visualizar os dados por meio das instruções SQL que utilizem o vínculo de Banco de Dados, podemos utilizar o código ”SELECT \* FROM emp@empresa\_exemplo.com;”.

Para simplificar nomes, pode ser utilizado o SYNONYM. Além de simplificar o acesso, referencia objetos de um outro usuário e deixa os nomes mais curtos. A sintaxe da instrução é representada por ”CREATE [PUBLIC] SYNONYM *synonym* FOR *object*;”.

SAIBA MAIS



Softwares diversos como firewalls que controlam o acesso a rede local ou programas antivírus que monitoram aplicações em execução no sistema operacional também têm papel fundamental nesse processo. Mas o software por si só não é cem por cento confiável e pode falhar, ter defeitos em sua implementação, ser atacado por invasões de *crackers* ou ataques de *hackers*.

Para contribuir para melhores níveis de segurança, podemos adicionar segurança física que envolva pessoas e infraestrutura para aumentar o nível de proteção dos sistemas do negócio em relação a outros males também prejudiciais à segurança desses sistemas.

Algumas medidas de prevenção que podemos elencar são: portas de segurança com acesso restrito, senhas para acesso a locais seguros como *data-centers* (centros de concentração de servidores e equipamentos de infraestrutura), treinamento preventivo e de combate a eventos e acidentes que possam prejudicar toda a infraestrutura e sistemas do negócio.

Fonte: os autores.

## GERENCIANDO SENHAS E RECURSOS

Caro (a) Aluno(a), também existe a possibilidade de gerenciar senhas utilizando *profiles*. A administração desses *profiles* é simples, controla a utilização de recursos por meio dos *profiles* e é possível ter informações sobre *profiles*, senhas e recursos.

*Profile* é um nome dado a um conjunto de limites de recursos e senhas. São anexados aos usuários por meio do comando *CREATE USER* ou *ALTER USER* e podem ser habilitados ou desabilitados.

O gerenciamento de senhas e perfis por meio do *profile* vai desde a criação do perfil, englobando os usuários, passando pelo controle de verificação de senha, expiração e validade de senhas, bloqueio de contas e histórico de senhas.

Para habilitar o gerenciamento de senhas ajuste o gerenciamento de senha utilizando *profiles* e anexando-o aos usuários. É possível bloquear, desbloquear e invalidar as contas de usuários utilizando o comando *CREATE USER* ou *ALTER USER*.

Ao bloquear usuários é possível utilizar alguns parâmetros pré-definidos, que são:

- FAILED\_LOGIN\_ATTEMPTS - Número de tentativas de *login* falhas antes de travar a conta do usuário.
- PASSWORD\_LOCK\_TIME - Número de dias que a conta ficará bloqueada.
- PASSWORD\_LIFE\_TIME - Tempo limite em dias para expirar a senha.
- PASSWORD\_GRACE\_TIME - Período de dias para a nova troca de senha após o primeiro *login* com sucesso.
- PASSWORD\_REUSE\_TIME - Número de dias antes de reutilizar uma senha novamente.
- PASSWORD\_REUSE\_MAX - Número máximo de vezes que a mesma senha poderá ser utilizada.
- PASSWORD\_VERIFY\_FUNCTION - Função *PL/SQL* para verificação da senha.

A função de verificação de senha deve validar o seguinte:

- O SYS deve ser o proprietário da função.
- Deve retornar *TRUE* ou *FALSE*.
- Deve seguir a assinatura padrão.
- Um erro na função causará erro no comando *CREATE* ou *ALTER*.

```
CREATE OR REPLACE FUNCTION VALIDA_SENHA (
    USER_ID IN VARCHAR2(30),
    PASSWORD_PARAMETER IN VARCHAR2(30),
    OLD_PASSWORD_PARAMETER IN VARCHAR2(30))
RETURN BOOLEAN;
```

A criação de um *profile* ou perfil, pode ser vista no exemplo a seguir:

```
CREATE PROFILE TEMPO_PARA_TROCA LIMIT
FAILED_LOGIN_ATTEMPTS 3
PASSWORD_LOCK_TIME UNLIMITED
PASSWORD_LIFE_TIME 30
PASSWORD_REUSE_TIME 30
PASSWORD_VERIFY_FUNCTION
valida_senha
PASSWORD_GRACE_TIME 5;
```

Por sua vez, a alteração de um perfil, pode ser vista no exemplo a seguir:

```
ALTER PROFILE default
FAILED_LOGIN_ATTEMPTS 3
PASSWORD_LIFE_TIME 60
PASSWORD_GRACE_TIME 10;
```

Observação: Menos de 1 dia:  $1/24 = 1$  hora

$$10/1400 = 10 \text{ minutos}$$

A exclusão de um *profile*, pode ser feita com instruções como “DROP PROFILE TEMPO;”, ou “DROP PROFILE TEMPO CASCADE;”, por exemplo.

No gerenciamento de recursos podem ser impostos limites de recursos para a sessão do usuário, por chamada ou ambos. Os limites podem ser definidos por meio de um *profile* e para habilitar a utilização de limites é só utilizar códigos como “RESOURCE\_LIMIT (*initSID.ora*);” ou “ALTER SYSTEM SET RESOURCE\_LIMIT=TRUE;”.

Os limites que podem ser observados no gerenciamento de recursos são:

- CPU\_PER\_SESSION - Total de tempo de *CPU* utilizada em centésimos de segundo.
- SESSIONS\_PER\_USER - Número de sessões concorrentes para o mesmo usuário.
- CONNECT\_TIME - Tempo decorrido em minutos.
- IDLE\_TIME - Período de inatividade.
- LOGICAL\_READS\_PER\_SESSION - Número de *data blocks* (leituras lógicas e físicas)
- PRIVATE\_SGA - Espaço privado na *SGA* em *bytes* (*Shared Server Only*).
- CPU\_PER\_CALL - Tempo de *CPU* em centésimos de segundo em cada chamada.
- LOGICAL\_READS\_PER\_CALL - Número de *data blocks* (leituras lógicas e físicas).

O exemplo da criação de um profile com a utilização de limites pode ser observado no código a seguir:

```
CREATE PROFILE developer_prof  LIMIT  
SESSIONS_PER_USER      2  
CPU_PER_SESSION        10000  
IDLE_TIME              60  
CONNECT_TIME            480;
```

As informações sobre senhas e limites de recursos podem ser obtidas no *data dictionary* (DD): *DBA\_USERS* e *DBA\_PROFILES*.

## GERENCIANDO USUÁRIOS

Ao gerenciar usuários é possível criar um novo usuário para a instância, bem como excluir e alterar as opções e monitorar suas informações. Os critérios de usuários e segurança norteiam alguns pontos importantes como, por exemplo, valores padrão de inicialização, privilégios e mecanismos de autenticação.

*Schema* é o nome dado para uma coleção de objetos. Ao criar um usuário, um *schema* correspondente ao usuário, também é criado. O usuário pode ser associado a somente um *schema* podendo conter objetos como gatilhos, sequências, visões, índices, tabelas ou sinônimos, por exemplo.

Para a criação de usuários, identifique a *tablespace* que o usuário usará para armazenar seus dados. Decida sobre as quotas na *tablespace default* e na temporária. Na sequência já é possível criar o usuário e garantir os seus privilégios necessários. Observe a seguir como se cria um usuário.

```
CREATE USER TESTE IDENTIFIED BY TESTE  
DEFAULT TABLESPACE USER  
TEMPORARY TABLESPACE TEMP  
QUOTA 15M ON USERS  
PASSWORD EXPIRE;
```

Na gerência do usuário é possível alterar um usuário usando comandos como “ALTER USER TESTE QUOTA 0 ON USERS;”, ou “ALTER USER TESTE IDENTIFIED BY XXI;”. Podemos também excluir um usuário usando o comando “DROP USER TESTE;”. Ter informações sobre o usuário por meio de comandos como “DBA\_USERS;” e “DBA\_TS\_USERS”.

## AUDITORIA

A auditoria consiste no monitoramento de um usuário e é usada para investigar atividades suspeitas no Banco de Dados, relatando informações sobre todas as atividades dos usuários.

Para uma boa auditoria, é necessário definir o que auditar, por exemplo: usuários, comandos ou objetos. Devem ser analisados os comandos executados com sucesso, sem sucesso ou ambos. A própria auditoria deve ser protegida de acesso não autorizado e ainda deve ser monitorado o tamanho e volume da auditoria, para que não prejudique o andamento do Banco de Dados. A auditoria é dividida em três categorias:

- Padrão - *Startup* e *shutdown* da instância e privilégios de administrador.
- Auditoria do Banco de Dados - Deve ser habilitada pelo DBA e não pode registrar valores para as colunas.

- Aplicação - Implementada por meio do código, pode armazenar valores para as colunas e usada para auditar trocas nas tabelas.

Uma auditoria da instância deve indicar no ***initSID.ora***, com a seguinte instrução: **AUDIT\_TRAIL=DB**. A gravação da auditoria ocorre na visão **SYS.AUD\$**, e seus comandos são:

- AUDIT TABLE.
- AUDIT CREATE ANY TRIGGER.
- AUDIT SELECT ON scott.emp.
- AUDIT DELETE ON scott.emp WHENEVER SUCCESSFUL.
- AUDIT INSERT on scott.emp WHENEVER NOT SUCCESSFUL BY ACCESS.
- AUDIT SELECT ON scott.emp BY SESSION.

A *package DBMS\_FGA* pode ser utilizada para refinar as opções de auditoria. E para remover a auditoria deve ser utilizado o comando: **NOAUDIT**. As visões do *data dictionary* são:

- ALL\_DEF\_AUDIT\_OPTS.
- DBA\_STMT\_AUDIT\_OPTS.
- DBA\_PRIV\_AUDIT\_OPTS.
- DBA\_OBJ\_AUDIT\_OPTS.
- DBA\_AUDIT\_TRAIL.
- DBA\_AUDIT\_EXISTS.
- DBA\_AUDIT\_OBJECT.
- DBA\_AUDIT\_SESSION.
- DBA\_AUDIT\_STATEMENT.

## CONSIDERAÇÕES FINAIS

Caro(a) aluno(a)! Com foco principal na Tecnologia da Informação, a segurança do Banco de Dados pode ser classificada em duas categorias distintas: segurança de sistema e segurança de dados. A segurança de sistema contém mecanismos de controle de acesso e o uso do Banco de Dados em um nível específico do sistema.

Os processos de segurança do sistema verificam a autorização para conexão ao Banco de Dados, auditoria do Banco de Dados e o que poderá ser executado por um usuário. Além disto, temos o controle de usuários e senhas, espaço disponível e limites de recursos dos usuários.

A segurança de dados inclui mecanismos de controle de acesso e o uso do Banco de Dados no nível de objeto de esquema incluindo os usuários com acesso ao objeto e os tipos específicos de ações que cada um pode executar.

Algumas ferramentas do *Oracle Server* adicionam um incremento à segurança, o que possibilita um ambiente multiplataforma de maior escalabilidade. É possível citar o *Oracle Enterprise Manager (OEM)*, e *Oracle Security Server Manager (OSS)*. O OEM é um conjunto de utilitários disponibilizados de forma gráfica em modo usuário (*GUI*), que permitem meios de gerenciar uma ou mais bases de dados. O OEM é composto por: Conjunto de ferramentas administrativas; Monitor de eventos; Agendador de tarefas; Interface gráfica para o Recovery Manager Tools.

Por sua vez o OSS pode ser utilizado para implementar uma estrutura mais complexa de segurança para dados mais sensíveis, com aspectos como Autenticação de usuário por meio de credenciais eletrônicas, Assinatura Digital, e *Single Sign On (SSO)*.

Por ser multiplataforma, a segurança não pode ser resguardada, com isso, a instalação do *Oracle* tem uma política que depende o mínimo do sistema operacional.

A primeira ação, é a alteração das senhas dos usuários padrão do banco. Usuários como *System /manager*, *Sys /change\_on\_install* e *DBSNMP /dbsnmp* que são instalados dessa forma e têm grande acesso ao banco, o que pode comprometer por completo a sua segurança.

## ATIVIDADES



1. O DBA é o profissional responsável pelo gerenciamento de um sistema de Banco de Dados, pela concessão de privilégios, pela liberação dos acessos e classificação dos usuários do sistema conforme as determinações das políticas de segurança. É ele quem cria usuários utilizando a instrução *CREATE USER*. Crie um usuário simplesmente atribuindo um nome de usuário *SCOTT* com a senha ou identificação *TIGER*.
  2. Quando o usuário é criado, o DBA pode conceder a ele privilégios específicos de sistema. Dê a concessão de privilégios de consulta na tabela *EMP* para o usuário criado no exercício 1:
  3. Os privilégios podem ser atribuídos conforme o objeto e suas possibilidades de acesso. Eles também podem variar de um objeto para outro. Leia atentamente as afirmações seguintes:
    - I. O proprietário de um objeto detém todos os privilégios sobre este objeto.
    - II. O proprietário de um objeto não tem permissão para conceder privilégios específicos sobre seus objetos para outros usuários.
    - III. O direito de um privilégio dado a um usuário pode ser retirado.
    - IV. É utilização da instrução *REMOVE* para revogar os privilégios concedidos a outros usuários.
    - V. Os privilégios concedidos a outros usuários por meio da cláusula *WITH GRANT OPTION* também serão revogados.
- Assinale a alternativa correta:
- a) Apenas I e II estão corretas.
  - b) Apenas II e III estão corretas.
  - c) Apenas I está correta.
  - d) Apenas II, III e IV estão corretas.
  - e) Apenas I, III e V estão corretas.

## ATIVIDADES



4. Existe a possibilidade de gerenciar senhas utilizando perfis. A administração desses perfis é simples, controla a utilização de recursos por meio deles e é possível ter informações sobre eles e suas respectivas senhas e recursos. Com base na ideia dos perfis, analise as afirmativas:

- I. *Profile* é um nome dado aos arquivos que contém o conjunto de usuários e senhas.
- II. Um *Profile* é anexado aos usuários por meio do comando *CREATE USER* ou *ALTER USER*.
- III. *Profile* pode ser habilitado ou desabilitado.
- IV. O gerenciamento de senhas e perfis por meio do *profile* atende somente a criação do perfil.
- V. As informações sobre *profile* podem ser obtidas por meio do data *dictionary DBA\_PROFILES*.

Assinale a alternativa correta:

- a) Apenas I, II e III estão corretas.
- b) Apenas II, III e IV estão corretas.
- c) Apenas I, IV e V estão corretas.
- d) Apenas II, III e V estão corretas.
- e) Apenas I, III e IV estão corretas.

5. A auditoria consiste no monitoramento de um usuário e pode ser usada para investigar atividades suspeitas no Banco de Dados, relatando informações sobre todas as atividades dos usuários. Sobre auditoria, assinale Verdadeiro (V) ou Falso (F):

- a) ( ) Para uma boa auditoria, é necessário definir o que auditar, por exemplo: usuários, comandos ou objetos.
- b) ( ) Na auditoria devem ser analisados somente os comandos executados sem sucesso.
- c) ( ) A própria auditoria deve ser protegida de acesso não autorizado.
- d) ( ) A própria auditoria deve ser monitorada o tamanho e volume da mesma, para que não prejudique o andamento do Banco de Dados.
- e) ( ) A Auditoria do Banco de Dados deve ser habilitada pelo DBA e não pode registrar valores para as colunas.



## Real Application Cluster (RAC)

Com a capacidade de oferecer altos índices de desempenho, tolerância a falhas e possibilidade de ajuste de acordo com a demanda (escalabilidade), esse recurso foi implementado em sua versão 10g, agora vem incluso com a aquisição de uma licença padrão (*standard*) do Oracle.

Nessa versão Standard existem limitações comuns em versões menos custosas como limite de computadores e recursos destes computadores, mas em termos de funcionalidade, seu ambiente mesmo nesta edição é muito bom.

Com o RAC, um Banco de Dados pode ser ajustado para funcionar o tempo todo ininterruptamente, sendo que em casos de manutenção preventiva planejadas ou corretiva devido a incidentes não previstos. Mesmo em casos de pane não prevista, instâncias fora da área afetada da infraestrutura podem continuar oferecendo os serviços de acesso ao Banco de Dados, mantendo um nível aceitável de eficiência (disponibilidade).

Assim, processos que ocorriam na parte afetada da infraestrutura podem migrar para instâncias funcionais de forma transparente, dando a impressão ao usuário de que nada esteja ocorrendo de grave na infraestrutura.

Essa escalabilidade transparente garantida por esse procedimento de migração de processos é muito importante para garantir a principal funcionalidade de um Banco de Dados, ainda mais com o recurso RAC.

Em grande parte dos casos, a solução utilizando RAC é mais eficiente, exceto em casos em que esse processamento em paralelo por meio de diversos servidores pode ser inferior.

Como exemplo, temos o caso de um Banco de Dados sendo executado em uma única instância tendo que distribuir seu processamento entre vários servidores, pode gerar atrasos nos processos devido a perda de desempenho no controle desse processamento distribuído.

O afunilamento natural que pode ocorrer em situações de pico de acesso a um Banco de Dados é minimizado devido ao recurso RAC distribuir as ações, da mesma forma que pode ocorrer no processamento de quantidades maiores de processos em sistemas OLTP (*Online Transaction Processing*).

Assim, é visível a melhora encontrada no uso desse recurso muito poderoso que acompanha o pacote Oracle.

Fonte: os autores.

# MATERIAL COMPLEMENTAR



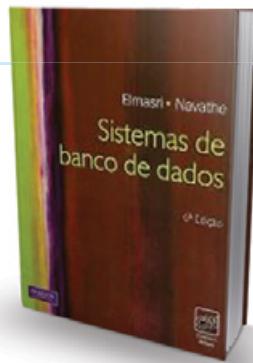
LIVRO

## Sistemas de Banco de Dados (2011)

Rames Elmasri e ShamKant Navathe.

**Editora:** Editora Pearson

**Sinopse:** a obra aborda conceitos fundamentais para proteger e usar os sistemas de Banco de Dados. Fundamentos de modelagem e de projeto de Banco de Dados. Linguagens e modelos fornecidos pelos sistemas de gerenciamento de Banco de Dados. Técnicas de implementação do sistema de Banco de Dados, com exemplos práticos.



# REFERÊNCIAS

WATSON, J. **OCA Oracle Database 11g:** administração I: guia do exame 1Z0-052. Porto Alegre: Bookman, 2010.

## Referências on-line

1- Em: <<http://docplayer.com.br/3158030-Banco-de-dados-oracle-10g-fundamentos-de-sql-ii.html>>. Acesse em: 28 nov. 2016.



## **GABARITO**

- 1) CREATE USER scott IDENTIFIED BY tiger;
- 2) GRANT select ON emp TO scott;
- 3) E. Apenas I, III e V estão corretas.
- 4) D. Apenas II, III e V estão corretas.
- 5) V - F - V - V - V.



# CONCLUSÃO

Prezado(a) aluno(a)! É com muita satisfação que apresentamos neste material alguns temas sobre Bancos de Dados. O conhecimento sobre bancos de dados e alguns de seus pontos são fundamentais para o desenvolvimento de softwares e tecnologias em inovação. Esse conhecimento faz toda a diferença na criação e administração dos dados e das infraestruturas necessárias.

Com esta abordagem é importante relembrar os pontos principais do material.

Na unidade I, abordamos inicialmente o conceito sobre a linguagem *Structured Query Language* (SQL), sendo que ela é a linguagem padrão utilizada nos banco de dados que seguem o modelo de dados relacional, para efetuar a manipulação dos dados em um banco de dados utilizamos os conceitos de *Doctrine Query Language* (DQL), *Data Manipulation Language* (DML), *Data Definition Language* (DDL), Data Control Language (DCL) e Data Transaction Language (DTL).

Por sua vez, na unidade II, conhecemos os formas de criação de um banco de dados no MySQL, além da criação, alteração e remoção das tabelas. Estudamos os comando de popular as tabelas. Nossa intenção foi fazer com que vocês acadêmicos vivenciarem na prática a manipulação de um Sistema de Gerenciamento de Banco de Dados (SGBD).

Na unidade III foi trabalhado conceitos mais avançados da linguagem SQL, sendo aplicada em um Banco de Dados. Falamos sobre Rollback e Commit. Também foram apresentados alguns comando especiais a serem utilizados em conjunto com a linguagem SQL.

Na unidade IV, vimos o conteúdo sobre Programação SQL. Foi utilizado o Banco de Dados Oracle, apresentado uma visão geral sobre Procedural Language/Structured Query Language (PL/SQL), além de tratar de *Procedures, Functions, Packages*.

Por fim a unidade V, abordou informações relevantes sobre *Triggers* e segurança do banco e dos dados. São tratados assuntos como o controle de acesso ao usuário, a concessão de privilégios, role ou regras, alteração de senha, privilégios de objeto, além de gerenciar senhas e recursos com a gerência dos próprios usuários e a auditoria realizada no banco de dados.

Esperamos ter alcançado nosso objetivo em passar nosso conhecimento a você, caro (a) aluno(a). Desejamos que você seja muito feliz ao percorrer o mundo profissional. Muito sucesso e paz!

