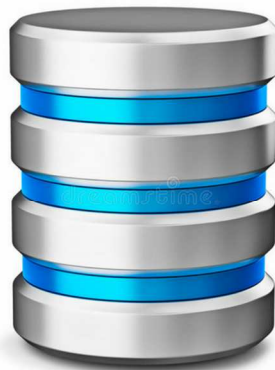


CURSO

SQL - ORACLE

**APRENDA O DOBRO
NA METADE DO TEMPO**



MICHEL BATISTA

SQL – aprenda o dobro na metade do tempo

1ª edição

São Paulo

Michel Batista Silva

2018

Copyright © 2018 Michel Batista

Todos os direitos são reservados e protegidos pela Lei 9610 de 19/02/1998.

*Esse livro está devidamente registrado na Biblioteca Nacional sob o número **ISBN: 978-85-924796-0-2***

Nenhuma parte deste livro poderá ser reproduzida ou transmitida por meios eletrônicos, fotográficos, gravação ou quaisquer outros sem autorização do autor.

O conteúdo desse produto se ampara no direito fundamental a manifestação do pensamento, previsto nos artigos. 5º, IV e 220 da Constituição Federal de 1988.

Vale-se o “animus narrandi”, protegido pela lei e pela jurisprudência AI nº 505.595, STF.

Site Oficial: www.foradserie.com.br

E-mail: michel@foradserie.com.br

Parabéns!

Você acaba de tomar uma importante decisão na busca de seus objetivos.

A decisão de investir em si mesmo!

Acredito profundamente que não exista fonte maior geradora de lucro, do que o investimento em conhecimento.

Me chamo Michel Batista, muito prazer!

Cursei Ciência da Computação em 2005 e tenho mais de 14 anos de experiência na área de sistemas, sempre trabalhando com o sistema de gestão integrada da Oracle, o Oracle Applications.

Não é de hoje que a área de sistemas é muito cobiçada.

É nela que está um dos melhores salários, junto com as melhores oportunidades do mercado.

Foi através da área de sistemas, que multipliquei em mais de 10 vezes meus ganhos, época em que ainda cursava a faculdade.

Já se foi o tempo em que trabalhar com sistemas era coisa de "Nerd", com aquela tela do monitor preta, letras cinza e linguagem "tecniquês".

A área de sistemas tem muito mais a oferecer do que apenas códigos e linguagens de programação.

Hoje em dia, é possível se especializar em negócios, processos, projetos, testes, etc.

Sou da época em que quando precisávamos de algum serviço bancário, era necessário chegar mais cedo ao banco, por volta de 9:15, aguardar em fila até as 10:00, hora em que se iniciava o expediente bancário.

Hoje movimentamos nossa conta bancária e pagamos nossos boletos através de um computador ou Smartphone, do conforto de nossas casas ou durante o percurso ao trabalho.

Atualmente, vemos cada vez mais serviços sendo informatizados.

Compra e recarga de créditos para transporte, vale alimentação (que antes eram em papel), ingressos de cinema, shows, jogos de futebol, são alguns outros exemplos de serviços que foram pouco a pouco sendo informatizados, levando praticidade e conforto a seus usuários e tornando a intervenção humana, cada vez mais dispensável.

Todos esses sistemas utilizam um banco de dados, onde a linguagem para extração e manipulação de dados, é o SQL.

A bola da vez agora, é a direção autônoma de carros.

A empresa norte americana Tesla e o Google estão derramando rios de dinheiro para tornar esse projeto possível e viável, em um futuro cada vez mais próximo.

Alguém duvida que logo teremos essa realidade em nossas vidas?

Quando toda essa revolução acontecer, de qual lado do tabuleiro você estará para jogar?

Do lado operacional, com sérios riscos de ter seu emprego extinto do mundo ou do lado que avança a passos largos rumo a inteligência artificial?

Tenho certeza que você estará do lado da revolução!

Espero que se identifique com esse curso e que esse material consiga te levar a outro nível, que seja capaz de transmitir a essência do que aprendi ao longo de todo esse tempo em que trabalho com sistemas.

Bons estudos e sucesso, sempre!

Sumário

Capítulo 1 - Introdução a Linguagem SQL com.....	10
Banco De Dados Oracle	10
<i>O que são SGBDs?.....</i>	<i>10</i>
<i>Um Pouco Sobre a História Da Linguagem SQL.....</i>	<i>10</i>
Capítulo 2 - Instalando Gratuitamente um Banco De Dados Oracle e SQL Developer.	12
Capítulo 3 - Introdução ao Modelo Entidade Relacionamento (MER).....	21
Entidades	21
<i>Tipo de relacionamento 1: Relacionamento 1 para 1.....</i>	<i>22</i>
<i>Tipo de relacionamento 2: Relacionamento 1 para vários.</i>	<i>22</i>
<i>Tipo de relacionamento 3: Relacionamento Vários para Vários.</i>	<i>23</i>
<i>Diagrama Entidade Relacionamento (DER).....</i>	<i>23</i>
Capítulo 4 – Introdução a Definição de Tabelas	25
Tipos De Dados	25
<i>Primary Key ou Chave Primária.....</i>	<i>27</i>
<i>Foreign Key ou Chave Estrangeira.....</i>	<i>27</i>
Constraints	28
Índices	32
Capítulo 5 - Introdução a Definição de Dados (DDL)	33
<i>Data Definition Language</i>	<i>33</i>
Capítulo 6 - Linguagem de Controle de Dados (DCL).....	34
<i>Data Control Language</i>	<i>34</i>
Capítulo 7 - Comandos De Manipulacao de dados (DML).....	35
<i>Data Manipulation Language (DML).....</i>	<i>35</i>
Capítulo 8 - Pratica em Comandos de Seleção.....	36
Select.....	36
<i>Inner Join.....</i>	<i>40</i>
<i>Outer Join.....</i>	<i>40</i>
<i>Right e Left Join</i>	<i>42</i>
<i>Alias.....</i>	<i>47</i>
Uso dos Comparadores.....	50
<i>Igualdade: Símbolo “ = “.....</i>	<i>51</i>
<i>Diferença: Símbolo “ <> ”</i>	<i>51</i>

Minoridade: Símbolo “ < ”	52
Maioridade: Símbolo “ > ”	53
Maioridade ou Igualdade: Símbolo “ >= ”	53
Minoridade ou Igualdade: Símbolo “ <= ”	54
Função Distinct	55
Operador Order By	56
Capítulo 9 – Operadores Condicionais	58
Operador IN e Not IN	58
Operador Between	59
Operador Like	60
Funções Count e Group By	62
Operadores Null e Not Null	64
Funções Exists e Not Exists	65
Outras Funções	67
Operador AND	67
Operador OR	69
Função Having	73
Having Sum	74
Having Max	75
Operador Union	77
Union All	78
Sub Query	80
Capítulo 10 – Prática em Comandos de Manipulação de Dados	84
Data Manipulation Language	84
Insert	84
Update	86
Delete	88
FOR UPDATE	90
Truncate	91
Capítulo 11 – Prática em Funções De Conversão de Dados.	93
Add_Months	93
Extract	94
Last Day	95
Months_Between	95
Next Day	96

Round	96
Trunc	97
Funções De Texto	98
ASCII	98
Concat	99
Initcap	101
Instr	101
Length	102
Lower	103
Upper	103
Lpad e Rpad	103
Funções Numéricas	108
ABS	108
AVG	109
Count	110
Greatest	111
Least	111
Max	112
Min	114
Mod	114
Outras Funções	117
Case	117
Comandos DCL Na Prática	122

Capítulo 1 - Introdução a Linguagem SQL com

Banco De Dados Oracle

O que são SGBDs?

SGBD (Sistema Gerenciador de Banco de Dados), é um sistema cuja função é armazenar dados em forma de tabelas, com a tarefa de manter os dados de forma íntegra, segura e de fácil acesso.

Um dos maiores e mais conhecido Gerenciador de Banco de Dados, é sem dúvida, o vendido pela empresa Oracle Corporation.

A Oracle produz sistemas gerenciadores de banco de dados há mais de 35 anos e seu banco de dados, é sem dúvida um dos mais avançados e seguros do mercado de SGBDs.

Um Pouco Sobre a História Da Linguagem SQL

O significado de SQL é “Structure Query Language” ou Linguagem de Consulta Estruturada.

Sua estrutura é declarativa e não em formato de linguagem de programação, o que faz com que sua linguagem seja inteligível a qualquer pessoa e não a apenas programadores de sistema.

O objetivo da linguagem SQL é fornecer as informações armazenadas no banco de dados, através de uma consulta em formato padrão.

O SQL foi desenvolvido no início dos anos 70, nos laboratórios da IBM em San Jose que tinha como objetivo, demonstrar a viabilidade da implementação do modelo relacional proposto pelo matemático britânico E. F. Codd.

O nome original da linguagem era SEQUEL, acrônimo para "Structured English Query Language" (Linguagem de Consulta Estruturada Inglesa).

Embora o SQL tenha sido originalmente criado pela IBM, rapidamente surgiram vários "dialetos" desenvolvidos por outros produtores.

Essa expansão levou à necessidade de ser criado e adotado, um padrão para a linguagem. Esta tarefa foi realizada pela American National Standards Institute (ANSI) em 1986 e ISO em 1987.

A linguagem SQL foi revista em 1992 e a esta versão, foi dado o nome de SQL-92. Foi revisto novamente em 1999 para se tornar SQL-99 (também chamado de SQL3) e em 2003, o SQL:2003.

No entanto, embora o SQL seja padronizado pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes de sistemas gerenciadores de bases de dados.

Tipicamente a linguagem pode ser migrada de plataforma para plataforma, sem grandes mudanças estruturais.

Capítulo 2 - Instalando Gratuitamente um Banco De Dados Oracle e SQL Developer.

1 - Criar uma conta gratuita no site oficial da Oracle:

<https://profile.oracle.com/myprofile/account/create-account.jspx>

2 - Escolha a versão de acordo com sua instalação do Windows ou Linux

Para Window 32 Bits:

<http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html>

Para Window 64 Bits:

<http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html>

Para Linux 64 Bits:

<http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html>

Aceite os termos de uso clicando na opção “Accept License Agreement”

☒ Accept License Agreement | ☐ Decline License Agreement

Escolha a versão de acordo com o seu sistema operacional:

Thank you for accepting the License Agreement; you may now download this software.

↓ Oracle Database Express Edition 11g Release 2 for Windows x64

- Unzip the download and run the DISK1/setup.exe

↓ Oracle Database Express Edition 11g Release 2 for Windows x32

- Unzip the download and run the DISK1/setup.exe

↓ Oracle Database Express Edition 11g Release 2 for Linux x64

-Unzip the download and the RPM file can be installed as normal

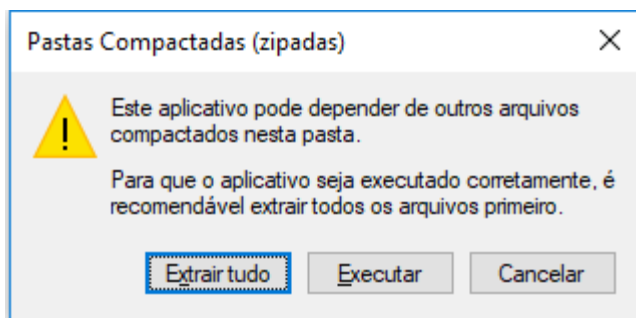
3- Após a conclusão do download, descompacte o arquivo, utilizando o software Winrar.

- Caso você não tenha o Winrar, baixe-o gratuitamente (por um determinado tempo) no site oficial:

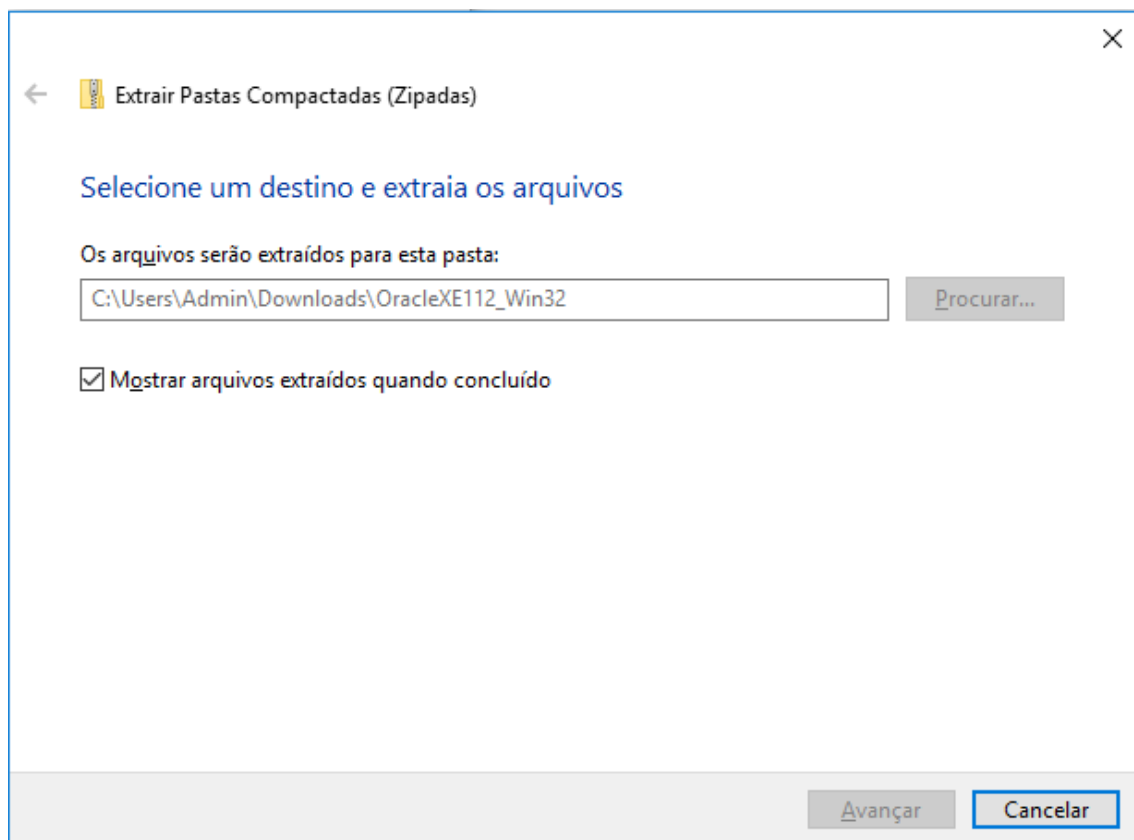
<http://www.winrarbrasil.com.br/winrar/download.mv>

4- Procure o arquivo de download OracleXE112 em seu computador.

- Execute o arquivo  setup.exe

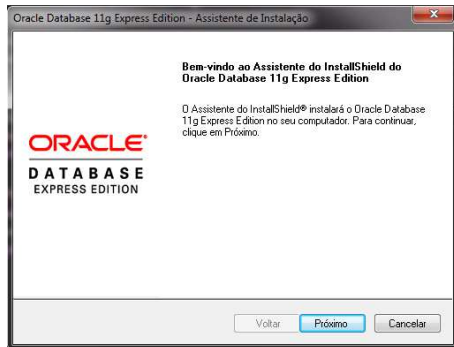


- Clique no botão Avançar.

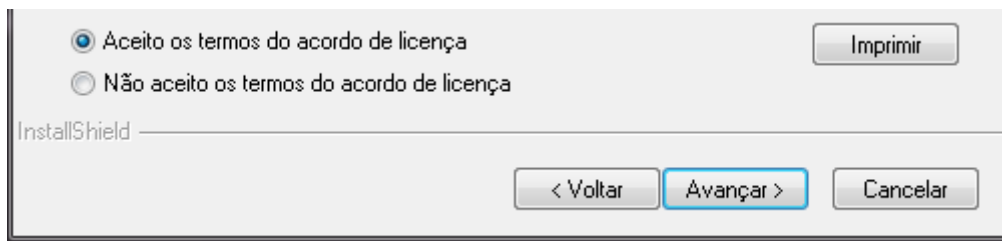


5- Abra a pasta Disk1 e em seguida execute o arquivo

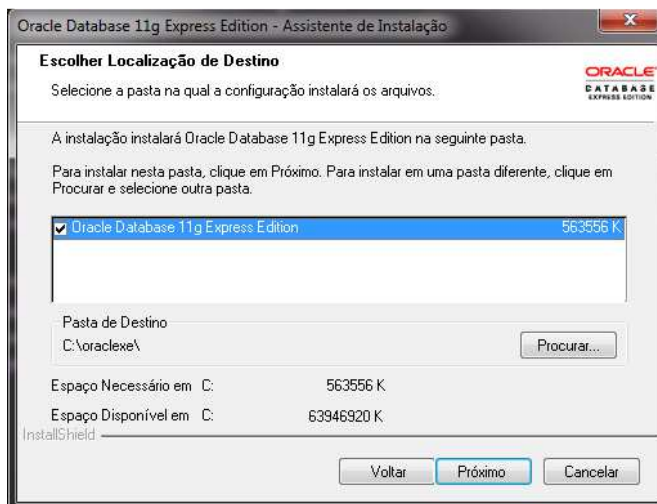
- Clique no botão “Próximo”



- Aceite os termos de licença e depois clique no botão “Avançar”.




- Defina a pasta onde o Banco de Dados Oracle será instalado e depois clique no botão Próximo.



6- Agora uma etapa muito importante.

- Defina a senha do usuário System do Banco de Dados e guarde-a com cuidado.
- Essa senha será usada para criar a primeira conexão com o banco de dados
- Depois clique no botão Próximo.



Oracle Database 11g Express Edition - Assistente de Instalação

Especifique Senhas de Bancos de Dados

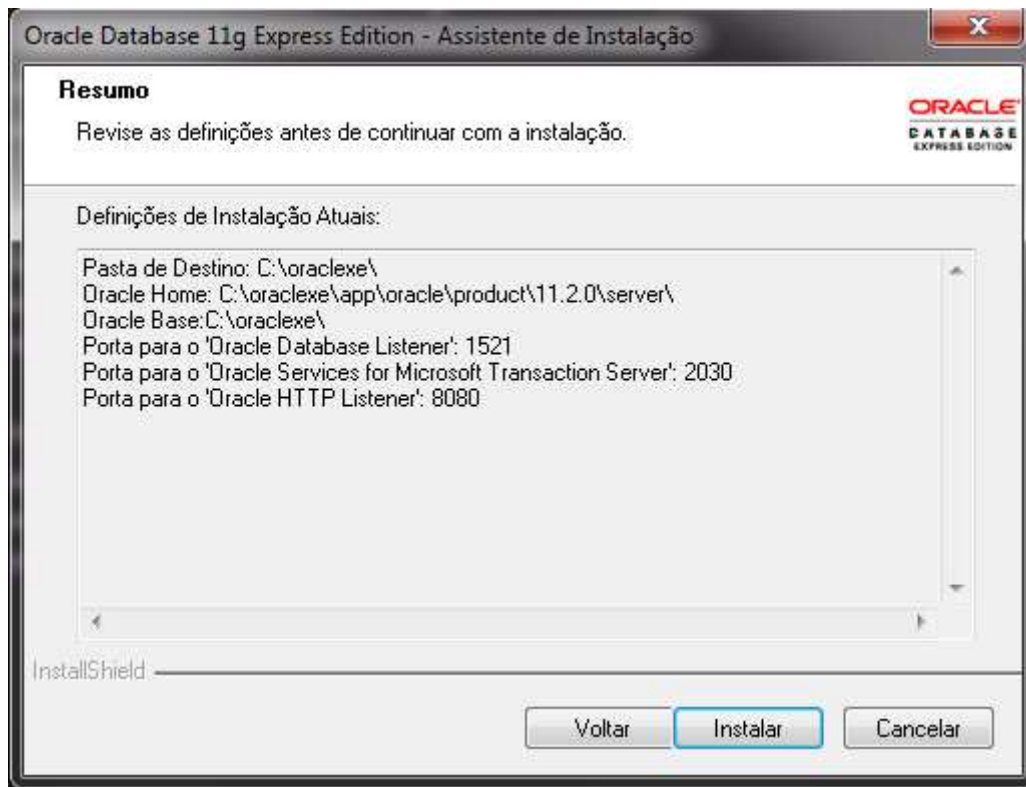
Informe e confirme as senhas do banco de dados. Essa senha será usada para as contas de bancos de dados SYS e SYSTEM.

Informar Senha

Confirmar Senha


InstallShield

- Revise as configurações e depois clique no botão Instalar.



7- Após o término da instalação, é a vez de instalar o SQL Developer, que é o software de interface onde executaremos as nossas consultas no banco de dados.

8- Agora precisamos instalar a versão gratuita do SQL Developer, da Oracle

 sqldeveloper-17.4.1.054.0712-no-jre






<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

- Escolha a versão de acordo com o seu sistema operacional:


SQL Developer 18.1

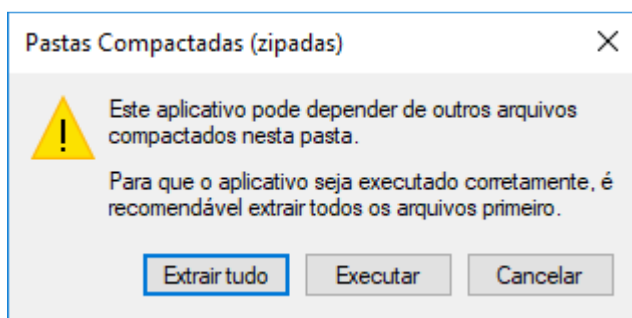
Version 18.1.0.095.1630 April 5, 2018

[New Features](#), [Release Notes](#), [Bugs Fixed](#), [Documentation](#)

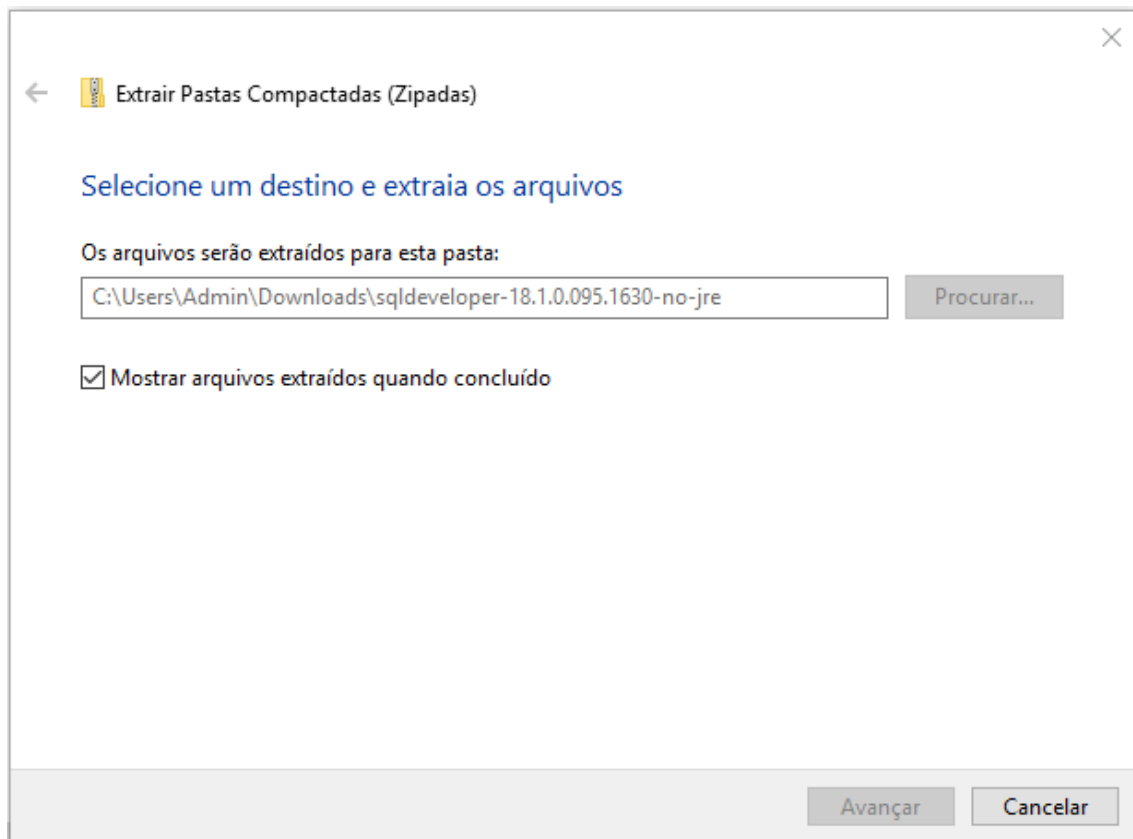
Windows 64-bit with JDK 8 included (2213a459d8915e724be7761661c2d640) Installation Notes	423 MB Download 
Windows 32-bit/64-bit (39b4e0ea8d3c6894e40089567e0ca367) Installation Notes , JDK 8 required	350 MB Download 
Mac OSX (9f77810a106f30cd34bba0e360119557) Installation Notes , JDK 8 required	350 MB Download 
Linux RPM (b4c8c705d04549b3ca74e65169bbc00b) Installation Notes , JDK 8 required	342 MB Download 
Other Platforms (39b4e0ea8d3c6894e40089567e0ca367)	350 MB Download 

9- Após o download do SQL Developer, descompacte-o em uma pasta em seu computador.

- Execute o arquivo  sqldeveloper
- Clique no botão extrair tudo:



- Escolha o diretório de destino e depois clique no botão Avançar:




10- Baixe o pacote de desenvolvimento JDK no site oficial da Oracle:

<http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>

- Aceite os termos de uso e faça o download do Java JDK 8 no site da Oracle de acordo com seu sistema operacional

Java SE Development Kit 8u172		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.99 MB	jdk-8u172-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.9 MB	jdk-8u172-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.07 MB	jdk-8u172-linux-i586.rpm
Linux x86	184.91 MB	jdk-8u172-linux-i586.tar.gz
Linux x64	167.15 MB	jdk-8u172-linux-x64.rpm
Linux x64	182.08 MB	jdk-8u172-linux-x64.tar.gz
Mac OS X x64	247.87 MB	jdk-8u172-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	140.05 MB	jdk-8u172-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.35 MB	jdk-8u172-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.63 MB	jdk-8u172-solaris-x64.tar.Z
Solaris x64	97.06 MB	jdk-8u172-solaris-x64.tar.gz
Windows x86	199.11 MB	jdk-8u172-windows-i586.exe
Windows x64	207.3 MB	jdk-8u172-windows-x64.exe

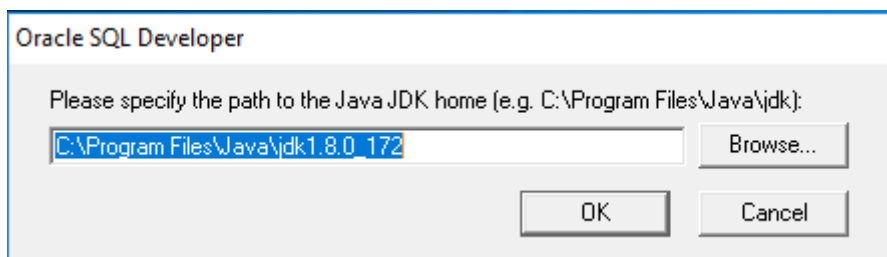
11- Execute o arquivo jdk-8u161-windows-i586

- Após a instalação do JDK, retorne a pasta de download do SQL Developer e crie um atalho na área de trabalho do arquivo  sqldeveloper

-No meu caso o download do SQL ficou no diretório:
C:\Users\Admin\Downloads\sqldeveloper-18.1.0.095.1630-no-jre\sqldeveloper

12- Agora abra o arquivo sqldeveloper

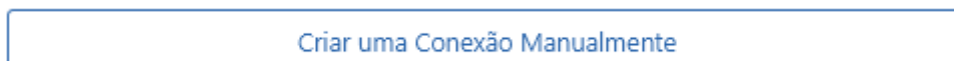
A janela abaixo será aberta. Se você tiver seguido esse passo a passo, o diretório do JDK será preenchido automaticamente, como na figura abaixo



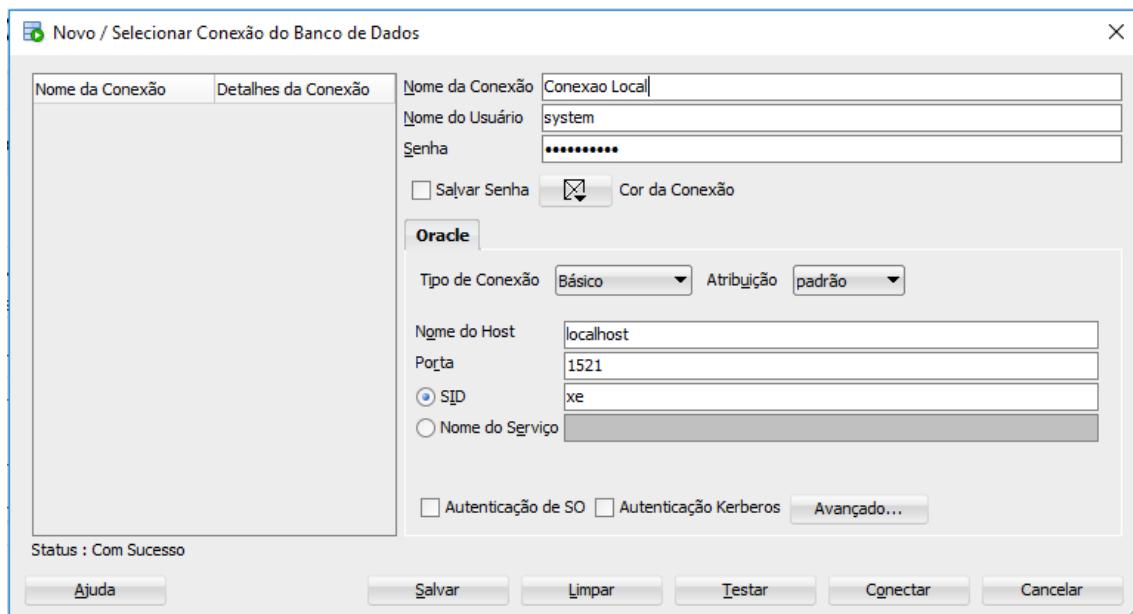
A janela do SQL Developer será aberta pela primeira vez



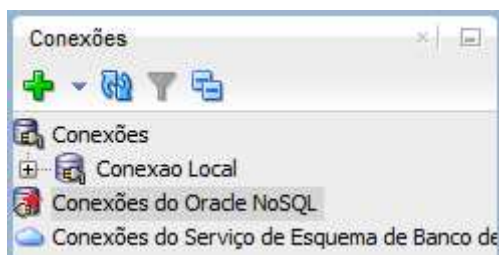
13- Clique no botão “Criar uma Conexão Manualmente”



- Preencha o campo “Nome da Conexão” com um nome de conexão de sua preferência.
- Nome do usuário: System
- Senha: Utilize a senha utilizada no passo 6
- Clique no botão “Conectar”



- Se tudo deu certo, uma nova conexão foi criada como nome que demos a ela. Nesse exemplo foi a “Conexao Local”



Capítulo 3 - Introdução ao Modelo Entidade Relacionamento (MER)

Antes de iniciarmos qualquer planejamento de dados, precisamos compreender conceitualmente o uso do MER e DER do sistema, que fará uso dos dados.

MER é um modelo que descreve as entidades (objetos) e características ou atributos do relacionamento.

Entidades

As entidades ou objetos, são as partes existentes e inexistentes do mundo real que serão transpostas para a estruturação dos objetos em um banco de dados.

Exemplo Prático: Um vendedor faz parte de uma entidade física, no caso uma empresa, pois ela existe fisicamente no mundo real.

A venda realizada pelo vendedor é uma entidade abstrata, já que faz parte do abstrato ou imaginário.

Agora que já sabemos como identificar os objetos do nosso modelo, é hora de definirmos como será seu relacionamento.

Tipo de relacionamento 1: Relacionamento 1 para 1

Esse é o tipo de relacionamento de mão única, ou seja, uma entidade se relaciona apenas com uma outra entidade.

Exemplo: Vamos definir uma entidade chamada de “Marido” e uma outra entidade que se chama “Esposa”.

Em um Projeto chamado “Casamento”, o relacionamento sempre será de exclusividade, onde um marido só pode se relacionar com apenas uma esposa e vice-versa.

Temos nesse caso, um relacionamento do tipo “1 para 1”.

Tipo de relacionamento 2: Relacionamento 1 para vários.

Esse é um tipo de relacionamento em que uma das entidades se relaciona com várias outras entidades.

Exemplo: Imagine que uma entidade de um relacionamento seja chamada de “MÃE” e uma outra entidade do relacionamento seja chamada de “FILHO”.

Uma mãe pode ter vários filhos. Temos nesse caso, um relacionamento do tipo 1 para vários ou também chamado de, “1 para N”.

Perceba que nesse caso, uma outra mãe não poderia ter os mesmos filhos que a primeira.

A entidade “FILHO” pode ser mais do que uma, mas a entidade mãe, é única.

Tipo de relacionamento 3: Relacionamento Vários para Vários.

Seguindo a mesma linha de raciocínio de nossos exemplos familiares, vamos assumir uma entidade chamada “TIO” e uma outra entidade chamada “SOBRINHO”.

Tios podem ter vários sobrinhos e vários sobrinhos podem ter vários tios.

Perceba que não existe exclusividade nesse tipo de relacionamento e temos então, um relacionamento de vários para vários.

Nesse tipo de relacionamento, temos o que chamamos de “N para N”.

Atributos Do Relacionamento

São as características que descrevem a entidade do relacionamento.

Por exemplo: Uma televisão possui um fabricante, código, modelo, tamanho, etc.

Essas características, são propriedades que identificam a entidade, podendo ser objetos ou pessoas.

Diagrama Entidade Relacionamento (DER)

É a representação gráfica do MER (Modelo Entidade Relacionamento).

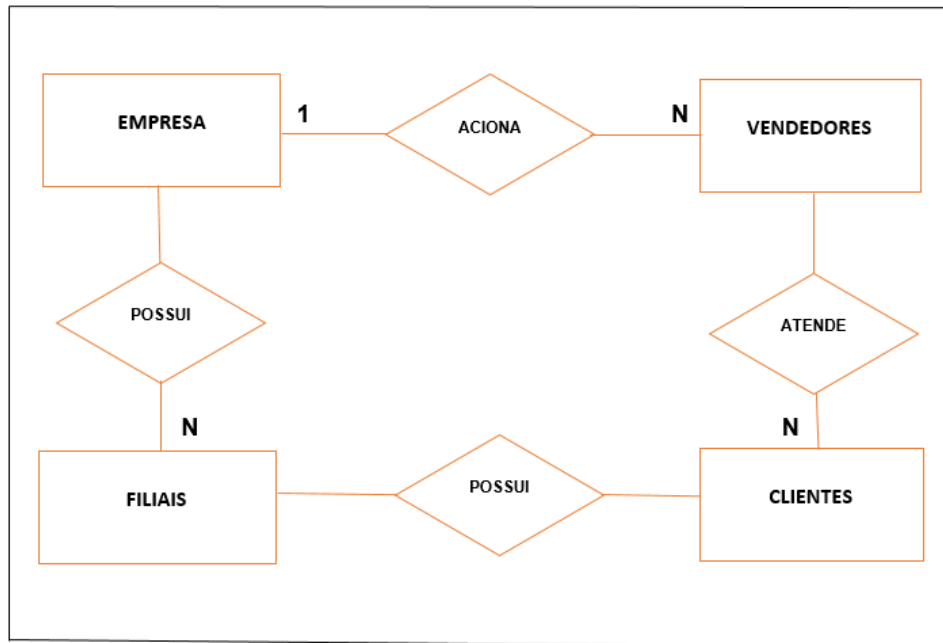
Por ser de característica visual, facilita o entendimento e discussão entre as equipes de análise e desenvolvimento do sistema.

As características do DER, são:

As entidades são representadas pelo símbolo retângulo.

Os atributos são representados por elipses

Os relacionamentos são representados por losangos.



Capítulo 4 – Introdução a Definição de Tabelas

Tipos De Dados

Ao criarmos uma tabela, consulta ou mesmo ao manipular dados, devemos nos atentar ao formato do tipo de dado, para que o banco de dados não retorne erros ao executar uma determinada tarefa.

Se estamos realizando uma operação aritmética, como soma ou multiplicação, devemos nos assegurar de que nenhuma coluna do tipo texto será retornada pela consulta, caso contrário, a pesquisa será executada com erro, já que estamos misturando letras com números, para realizar uma operação aritmética.

Não podemos comparar datas com números, nem números com textos.

Para realizarmos esse tipo de tarefa, devemos converter uma das partes, para que ambos fiquem com o mesmo formato de dados.

Vamos iniciar, criando uma tabela com informações básicas de um cliente.

Não se preocupe com os comandos apresentados, pois falaremos sobre eles no tópico de DDL (Data Definition Language).

```
CREATE TABLE CLIENTES
( CODIGO_CLIENTE NUMBER(15) NOT NULL
, COD_FILIAL      NUMBER(15)
, NOME_CLIENTE   VARCHAR(240) NOT NULL
, CNPJ           NUMBER(14) NOT NULL
);
```

As colunas “Codigo_Cliente”, “Cod_Filial”, “Nome_Cliente” e “CNPJ” serão as colunas de nossa tabela chamada de “Clientes”.

Estamos definindo as colunas “codigo_cliente” e “cod_filial” com formato numérico de 15 posições.

A coluna “cnpj” também será do tipo numérica, mas com 14 posições.

O sistema que utilizar essa tabela, poderá gravar um código de cliente e filial com até 15 números, nas respectivas colunas “código_cliente” e “cod_filial” e não poderá deixar a coluna “código_cliente” sem nenhuma informação.

O preenchimento da coluna “cnpj”, também será obrigatória e deverá ser preenchida com no máximo, 14 posições.

Não será aceito nenhum outro tipo de caractere que não seja numérico.

Já a coluna “nome_cliente” é do tipo VARCHAR, que é do tipo texto ou string.

Seu preenchimento também é obrigatório e limitado a 240 posições.



Note que na coluna “Cod_Filial” não possui a instrução **NOT NULL**.

Isso significa que no momento em que cadastrarmos um cliente, essa informação poderá ficar vazia ou nula.

Nas demais colunas, todas as informações deverão obrigatoriamente ser preenchidas, para que seja possível gravar o cadastro completo do cliente com sucesso na respectiva tabela do banco de dados.

Agora vamos criar uma tabela de endereços, que a chamaremos de “Endereco_Clientes”:

```
CREATE TABLE ENDERECO_CLIENTES
( CODIGO_CLIENTE NUMBER(15) NOT NULL
, COD_FILIAL      NUMBER(15) NOT NULL
, ENDEREÇO        VARCHAR(240) NOT NULL
, NUMERO          NUMBER(10) NOT NULL
, COMPLEMENTO     VARCHAR(240)
, ESTADO          VARCHAR(2) NOT NULL
, CIDADE          VARCHAR(50) NOT NULL
, TELEFONE        NUMBER(11)
);
```

Note que essa tabela também possui uma coluna chamada “Codigo_Cliente”.

Essa coluna será o elo de ligação entre a nossa tabela de endereços “Endereco_Clientes” e a tabela de clientes “Clientes”.

Definimos então, que a coluna “codigo_cliente” será a chave primária da tabela “Clientes” e será a chave estrangeira da tabela “Endereco_Clientes”.

Primary Key ou Chave Primária

Via de regra, é uma chave de composição numérica, que garante a unicidade ou exclusividade da entidade no relacionamento.

Um exemplo prático: Imagine que um hipotético cliente defina que a chave primária de sua tabela de clientes, será o CNPJ.

Com essa estratégia, ele garante uma chave única na tabela de clientes, visto que nenhum outro cliente terá o mesmo número, já que o número do CNPJ é um valor único e exclusivo, onde cada cliente possui o seu.

Foreign Key ou Chave Estrangeira

De uma forma didática, é uma coluna (ou mais) contida na tabela filho, que forma um elo de ligação com a tabela pai, por meio de uma chave única.

Essa chave de ligação, será a responsável por estabelecer um relacionamento de duas ou mais tabelas, se conectando por meio de uma ou mais colunas.

Veja que em nosso exemplo, temos o relacionamento do tipo “1 para N”, onde um cliente pode ter vários endereços.

Foi exatamente isso que aprendemos no capítulo três, “Modelo Entidade Relacionamento” desse curso.

Constraints

Constraints são restrições aplicadas às tabelas do banco de dados, com o objetivo de manter a integridade dos dados.

Observe o exemplo abaixo:

Vamos imaginar um projeto onde no modelo lógico, definimos que o relacionamento entre a entidade “Cliente” e a entidade “Endereço” seja de “1 para N”, ou seja, um único cliente poderá ter vários endereços.

O sistema que utiliza o banco de dados desse projeto, permitiu que um endereço fosse cadastrado, utilizando um código de cliente que não existe na tabela de clientes.

Tabela de Clientes

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ
1	10	1	FILIAL INDUSTRIA ...	12345678912345

Tabela de Endereços

	CODIGO_CLIENTE	COD_FILIAL	ENDEREÇO	NUMERO	COMPLEMENTO	ESTADO	CIDADE	TELEFONE
1	99999	10	AV PAULISTA ...	1		SP	SAO PAULO ...	11987654321

Como é possível o banco de dados que é íntegro, ter permitido que um registro na tabela de endereços fosse inserido, sem que esse mesmo código já não existisse antes na tabela de clientes?

Isso aconteceu, porque essa definição do “MER” de “1 para N” existe apenas no modelo lógico, ou seja, por serem entidades abstratas, não foram implantadas no banco de dados, para impedir que um código inexistente da tabela pai, seja inserido na tabela filha.

Uma das formas de impor esse tipo de restrição, é através do uso de “Constraints”.

Constraints são restrições, que asseguram que a integridade do modelo lógico, seja aplicada no modelo físico, no banco de dados.

As “Constraints” impedem que um registro filho seja criado sem que o registro pai exista ou que um registro pai seja excluído, quando houver registros na tabela dependente (filhos).

Agora vamos recriar nossas tabelas de “Clientes” e “Endereços”, com as chaves primárias e estrangeiras, utilizando as respectivas Constraints necessárias, para garantir a integridade dos dados, definidas no modelo lógico.

Veja abaixo a tabela “Clientes” recriada , agora com a chave de Primary Key no campo “Codigo_Cliente”.

```
CREATE TABLE CLIENTES
( CODIGO_CLIENTE NUMBER(15) NOT NULL
, COD_FILIAL NUMBER(15)
, NOME_CLIENTE VARCHAR(240) NOT NULL
, CNPJ NUMBER(14) NOT NULL
, CONSTRAINT PK_CODIGO_CLIENTE PRIMARY KEY (CODIGO_CLIENTE)
);
```

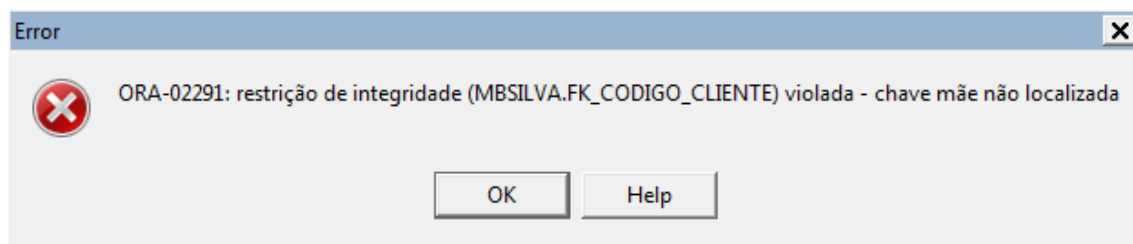
Agora é a vez da tabela “Endereco_Clientes”, com a respectiva chave de Foreign Key, fazendo referência ao campo “codigo_cliente”, da tabela “Clientes”.

```

CREATE TABLE ENDereco_CLIENTES
( CODIGO_CLIENTE NUMBER(15) NOT NULL
, COD_FILIAL      NUMBER(15) NOT NULL
, ENDEREÇO        VARCHAR(240) NOT NULL
, NUMERO          NUMBER(10) NOT NULL
, COMPLEMENTO     VARCHAR(240)
, ESTADO          VARCHAR(2) NOT NULL
, CIDADE          VARCHAR(50) NOT NULL
, TELEFONE        NUMBER(11),
CONSTRAINT FK_CODIGO_CLIENTE
FOREIGN KEY (CODIGO_CLIENTE)
REFERENCES CLIENTES (CODIGO_CLIENTE)
);

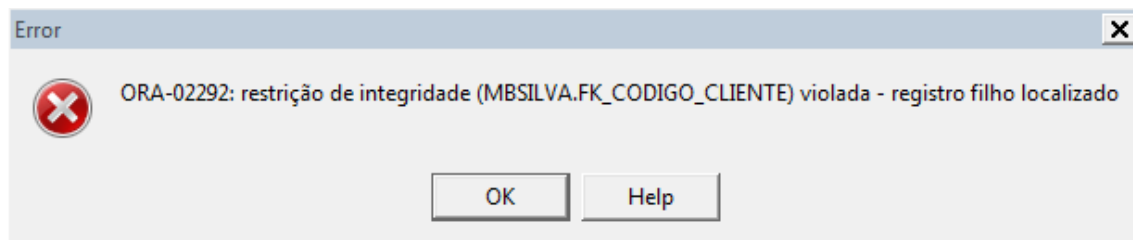
```

Vamos repetir o mesmo cenário de testes, incluindo um registro na tabela “Endereco_Clientes”, sem que o respectivo registro pai exista na tabela “Clientes”.



Resultado: Dessa vez, o banco de dados impediu que um registro filho fosse criado, sem que houvesse um registro pai (ou mãe) criado antes.

Agora, vamos tentar deletar o registro pai, sem antes deletar o registro filho.



Resultado: O banco de dados não permitiu a exclusão, já que estamos tentando deletar um registro pai da tabela “Clientes”, onde existem registros filho associados na tabela “Endereco_Clientes”.

Diferentemente do modelo lógico, temos agora uma restrição criada diretamente no banco de dados, através de um objeto do tipo "Constraint".

Veja que o Oracle EBS (Enterprise Business Suite), que é o sistema de gestão empresarial da própria Oracle, não utiliza Constraint em seu modelo de dados.

Nesse caso, o próprio software do Oracle EBS garante a integridade dos dados.

Mas não seria mais seguro, garantir a integridade do modelo de dados através do uso de Constraints?

A resposta é: Depende!

O uso de Constraint demanda mais recursos do banco de dados, uma vez que haverá um passo a mais a ser validado, antes que a efetivação da transação seja efetivamente realizada.

Em uma empresa varejista, onde milhões de registros transacionais são gerados diariamente, o uso de Constraint pode resultar em uma queda de performance do sistema e com isso, maior tempo de processamento das transações.

A tabela do banco de dados onde ficam gravadas as Constraints, se chama DBA_CONSTRAINTS.

Vamos consultar na tabela , a Constraint FK_CODIGO_CLIENTE que acabamos de criar.

```
SELECT * FROM dba_constraints  
WHERE constraint_name = 'FK_CODIGO_CLIENTE'
```

OWNER	CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME	SEARCH_CONDITION	R_OWNER	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
MBSILVA	FK_CODIGO_CLIENTE	R	ENDEREÇO_CLIENTES	<Long>	MBSILVA	PK_CODIGO_CLIENTE	NO ACTION	ENABLED

Índices

Uma outra forma de garantir a unicidade de registros em uma tabela, é através do uso de índices do tipo “Unique”.

A sintaxe para criação é:

```
CREATE [ tipo de índice opcional ] INDEX [ nome do índice ] ON [ tabela ] ( [coluna] )
```

Índices funcionam exatamente como o índice de um livro.

Em livros, a função do índice é localizar com maior rapidez uma determinada página ou capítulo.

É dessa mesma forma, que o banco de dados localiza com maior rapidez os registros (linhas) solicitados pela pesquisa.

Ao utilizarmos índices do tipo “Unique”, o banco de dados não permite que a chave definida no índice, que na verdade é uma coluna de uma tabela, seja duplicada durante a inserção ou atualização do registro (linha).

Vamos criar um índice chamado “cod_func” em uma tabela chamada “salarios”

```
CREATE UNIQUE INDEX cod_func ON salarios (cod_funcionario)
```



Índices do tipo “Unique” garantem apenas a unicidade de registros na tabela onde está sendo criado, não realizando nenhum tipo de validação de chave primária e estrangeira, como acontece com o uso de Constraints.

Capítulo 5 - Introdução a Definição de Dados (DDL)

Data Definition Language

O significado de “Data Definition Language”, é *Linguagem de Definição de Dados*, em português.

É um grupo de comandos da linguagem SQL usado para definir a estrutura de dados como, criar, modificar e remover objetos de um banco de dados.

Os principais comandos DDL são:

CREATE: É o comando usado para criação de objetos no banco de dados como: Tabelas, Views, Triggers, Packages, Procedure, Functions, etc.

ALTER: É o comando utilizado para alteração da estrutura de tabelas, sessão do usuário, etc.

DROP: É o comando utilizado para deleção de objetos no banco de dados como: Tabelas, Triggers, Packages, Procedure, Functions, etc.

Capítulo 6 - Linguagem de Controle de Dados (DCL)

Data Control Language

O termo “Data Control Language” significa, Linguagem de Controle de Dados, em português.

É o grupo de comandos utilizado pelo administrador de dados, para conceder ou remover permissões em objetos do banco de dados.

Seus principais comandos, são:

GRANT: Comando utilizado para conceder permissão de acesso a objetos de outros usuários.

As permissões podem ser de Inserção, Atualização ou Deleção para objetos do tipo Tabela ou de Execução para objetos do tipo Procedure ou Package.

REVOKE: Comando utilizado para remover as permissões de acesso a objetos, concedidas previamente a um usuário do banco de dados.

Para colocar nosso conteúdo em prática dos exemplos desse curso, será necessário que você já tenha instalado em seu computador, um banco de dados Oracle e um Client de interface com o banco de dados.

É possível utilizar o software SQL PLUS, que é instalado automaticamente juntamente com o banco de dados Oracle, mas esse software é indicado a usuários avançados.

Capítulo 7 - Comandos De Manipulacao de dados (DML)

O termo “Data Manipulation Language” significa, Linguagem de Manipulação de dados, em português.

Para facilitar a didática do nosso curso, sugiro que instale o software PL/SQL Developer, TOAD ou o SQL Developer, que além de ser gratuito, é fornecido pela própria Oracle.

Client é o software de interface, que conecta o usuário ao banco de dados.

Esse software, fará o envio de nossas instruções SQL e receberá as informações de retorno do banco de dados.

Nos exemplos fornecidos em nosso curso, utilizamos o Banco de Dados Oracle e o PL/SQL Developer, como software client de interface.

Data Manipulation Language (DML)

É o grupo de instruções da linguagem SQL utilizados para manipulação de informações gravadas em tabelas do banco de dados.

INSERT: É o comando utilizado para inserir registros (linhas) em tabelas do banco de dados.

UPDATE: É o comando utilizado para atualizar registros (linhas) que estão armazenados em tabelas do banco de dados.

DELETAR: É o comando utilizado para excluir registros (linhas) que estão armazenados em tabelas do banco de dados.

Capítulo 8 - Prática em Comandos de Seleção

Entre os profissionais que trabalham com SQL, o termo mais comum quando falamos em pesquisa ou consulta ao banco de dados, é o termo em inglês chamado “Query”.

Nos capítulos anteriores, vimos um pouco da teoria sobre comandos da linguagem SQL.

Agora vamos a parte prática.

Select

Select : *É o comando SQL utilizado para extração dos dados de uma ou mais tabelas do banco de dados.*

Asterísco (*): *É o comando SQL “coringa” do banco de dados.*

Ao utilizar o asterisco na pesquisa, indicamos ao banco de dados que queremos que seja retornada pela consulta, todas as colunas de todas as fontes de dados.


From: *É a instrução SQL utilizada para declarar as fontes de dados da pesquisa.*

As fontes de dados podem ser Tabelas, Views ou Sub Consultas.

Where: *É a instrução SQL que impõe uma ou mais condições para o retorno de uma pesquisa.*

Exemplo:

```
SELECT *  
  FROM clientes  
 WHERE cod_filial = 10
```



	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	CLIENTE FILIAL ...	98765432198765	Y
2	8888	10	CLIENTE TESTE ...	12345678912345	N
3	7777	20	CLIENTE 7777 ...	68719871897878	Y

Na consulta acima, desejamos que todas as colunas da tabela “Clientes” sejam exibidas (pelo uso do asterisco) Onde o código da filial seja igual a 10.

Existem dois tipos de sintaxe:

Uma delas, é utilizando o operador condicional “WHERE” juntamente com o operador igual (=) para junção de tabelas.

```
SELECT *  
  FROM clientes      cli  
    ,endereco_clientes end_cli  
 WHERE cli.codigo_cliente = end_cli.codigo_cliente
```

SELECT [nome do campo] FROM [tabela] WHERE [condição] = [condição]

Recapitulando:

SELECT: É a instrução SQL usada para selecionar uma ou mais colunas de tabelas do banco de dados.

No exemplo acima, não utilizamos nenhuma coluna específica para exibição e sim, o símbolo de asterisco (*).

O uso do símbolo de asterisco (*), sinaliza ao banco de dados que desejamos que todas as colunas da (s) tabela (s) sejam exibidas.

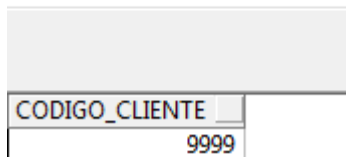
FROM: É a instrução da linguagem SQL usada para que seja declarada as fontes de dados à consulta.

WHERE [opcional]: É o operador condicional que estabelece regras para que a realização da consulta seja efetuada.



Quando trabalhamos com dados do tipo Data, Char e Varchar (que são do tipo Texto ou String), sempre devemos declará-las dentro de intervalos de aspas simples (' ').

```
SELECT codigo_cliente
FROM clientes
WHERE status = 'Y'
```

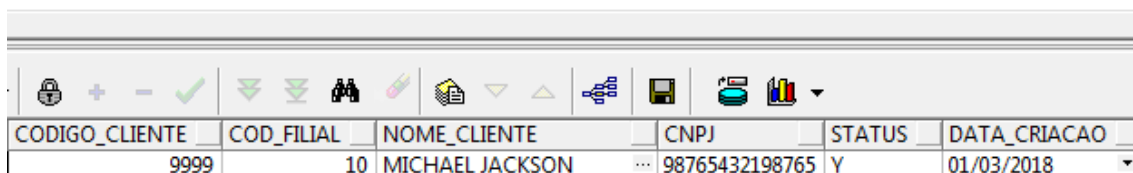


CODIGO_CLIENTE
9999

Veja que o campo status é do tipo texto e por isso, ao compararmos seu conteúdo a letra “Y”, tivemos que declará-lo dentro de aspas simples (' ').

Quando desejamos fazer um comentário em uma consulta, seja para um lembrete ou uma anotação qualquer, usamos dois traços (--) para que esse comentário seja ignorado pelo interpretador SQL.

```
SELECT * -- Essa query retornará todos os registros
FROM clientes
```



CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018

Quando desejamos comentar um bloco de uma consulta, colocamos o comentário dentro de um intervalo (/* */).

```
SELECT codigo_cliente
      /*, cod_filial
      , nome_cliente*/
FROM clientes
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018

Perceba que o interpretador SQL reconheceu somente o termo “codigo_cliente” na consulta, pois as colunas “cod_filial” e “nome_cliente” foram “comentadas”, por estarem dentro de um intervalo do tipo /* */

O outro tipo de sintaxe, é feita através da declaração dos operadores INNER, LEFT, RIGHT e FULL JOIN para também representar a junção de tabelas.

```
SELECT [ nome do campo ]
FROM [ tabela ] ( INNER, LEFT, RIGHT, FULL ) JOIN [ tabela ]
ON [ tabela.coluna ] = [ tabela.coluna ]
```

```
SELECT *
FROM clientes
INNER
JOIN endereco_clientes
ON clientes.codigo_cliente = endereco_clientes.codigo_cliente
```

INNER, LEFT, RIGHT e FULL JOIN: Especifica a tabela a ser ligada e qual será o tipo de junção à outra tabela.



Ao longo dos mais de 14 anos em que trabalho em projetos Oracle, vejo que a maioria absoluta dos programadores, utilizam a sintaxe do primeiro exemplo, devido a sua praticidade e simplicidade de compreensão da consulta.

É claro que nada disso impede em você ter mais afinidade com a construção de consultas, através da declaração explícita do operador “Join”.

Inner Join

Ao unirmos duas ou mais tabelas através de uma igualdade de chaves, estamos realizando uma operação do tipo “Inner Join”.

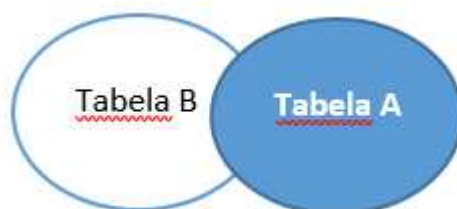
No nosso caso, estamos unindo as tabelas “Clientes” e “Endereco_Clientes”, através da chave “código_cliente”.

Tecnicamente falando, estamos realizando um “Inner Join”.

Outer Join

O “Outer Join” é utilizado quando unimos duas ou mais tabelas através de suas respectivas chaves, porém, em uma das tabelas, serão retornados todos os seus registros definidos pela pesquisa.

Em termos práticos, utilizamos o “Outer Join” quando não temos certeza se todas as chaves da tabela “A” estarão contidas na tabela “B” ou quando desejamos que todos os registros da tabela “A” sejam exibidos.



Exemplo:

Vamos realizar uma pesquisa simples na tabela de clientes:

```
SELECT * FROM clientes
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
8888	10	MICAELA INDUSTRIA	12345678912345	N	02/03/2018
3333	20	DML COMERCIO	78765432198765	Y	03/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
5555		CLINT GLACIAL	68719871897878	Y	06/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Resultado: Temos 7 registros

Agora, vamos realizar uma pesquisa simples na tabela de endereços:

```
SELECT * FROM endereco_clientes
```

CODIGO_CLIENTE	COD_FILIAL	ENDEREÇO	NUMERO	COMPLEMENTO	ESTADO	CIDADE	TELEFONE	STATUS
9999	10	AV PAULISTA	1		SP	SAO PAULO	1139751423	Y
8888	20	AV SAO JOAO	10		SP	SAO PAULO	1125431524	Y
7777	30	AV RIO GUARULHOS	123		SP	GUARULHOS	1145432154	Y

Resultado: Temos 3 registros.

Temos então 7 registros na tabela clientes e 3 registros na tabela de endereços.

Agora vamos unir as duas tabelas através da chave “codigo_cliente”

```

SELECT cli.codigo_cliente
      , cli.nome_cliente
      , end_cli.endereço
FROM clientes      cli
      , endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente

```

CODIGO_CLIENTE	NOME_CLIENTE	ENDEREÇO											
9999	MICHAEL JACKSON	AV PAULISTA											
8888	MICAELA INDUSTRIA	AV SAO JOAO											
7777	MICAELA FILIAL	AV RIO GUARULHOS											

Perceba que estamos realizando a pesquisa, usando uma junção do tipo “Inner Join”, onde buscamos por uma correspondência exata do mesmo “codigo_cliente” em ambas as tabelas, clientes e endereços.

Então, a pesquisa só nos retornou 3 registros, pois a tabela de endereços não possui os códigos de cliente 3333, 4444, 5555 e 6666.

E se quiséssemos então retornar todos os clientes, mesmo os registros onde não há o código de cliente na tabela de endereços?

Precisamos então, alterar o tipo de pesquisa, substituindo conceitualmente o “Inner Join” por um “Outer Join”.

Right e Left Join

Ao utilizarmos a junção pelo método “Outer Join”, precisamos informar ao banco de dados, em qual tabela não há correspondência exata de chave.

Voltando ao nosso exemplo anterior, na tabela de endereços não há registros correspondente aos códigos 3333, 4444, 5555 e 6666.

Faremos então um “Outer Join” pela direita (“Right Outer Join”):

```

SELECT cli.codigo_cliente
      , cli.nome_cliente
      , end_cli.endereço
FROM clientes      cli
      , endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente(+)

```

CODIGO_CLIENTE	NOME_CLIENTE	ENDEREÇO
9999	MICHAEL JACKSON	AV PAULISTA
8888	MICAELA INDUSTRIA	AV SAO JOAO
7777	MICAELA FILIAL	AV RIO GUARULHOS
5555	CLINT GLACIAL	
6666	INDUSTRIA SORRIDENTE	
4444	COMERCIO DA ESQUINA	
3333	DML COMERCIO	

Percebeu o símbolo (+) ao lado de “end_cli.codigo_cliente” ?

O sinal (+) é o comando que sinaliza ao banco de dados, que estamos realizando uma junção de tabelas do tipo “Outer Join”.

Como a tabela/coluna “end_cli.codigo_cliente” (referente ao endereço) está declarada a direita na comparação de igualdade, estamos então realizando um “Right Outer Join”.

E se a tabela/coluna “end_cli.codigo_cliente” estivesse declarada à esquerda na comparação de igualdade, como no exemplo abaixo ?

Estaríamos fazendo então, um “Left Outer Join”

```

SELECT cli.codigo_cliente
      , cli.nome_cliente
      , end_cli.endereço
FROM clientes      cli
      , endereco_clientes end_cli
WHERE end_cli.codigo_cliente(+) = cli.codigo_cliente

```

CODIGO_CLIENTE	NOME_CLIENTE	ENDEREÇO
9999	MICHAEL JACKSON	AV PAULISTA
8888	MICAELA INDUSTRIA	AV SAO JOAO
7777	MICAELA FILIAL	AV RIO GUARULHOS
5555	CLINT GLACIAL	
6666	INDUSTRIA SORRIDENTE	
4444	COMERCIO DA ESQUINA	
3333	DML COMERCIO	

Resultado: Ao utilizarmos uma junção pelo método “Outer Join” (tanto Right quanto Left), foi possível retornar **todos os registros** da tabela de clientes e retornar também seus respectivos endereços, onde houve correspondência exata do código de cliente na tabela de endereços.

Onde não houve correspondência exata do código de cliente na tabela de endereços, a coluna “Endereco” ficou vazia, já que não existe os códigos 3333, 4444, 5555 e 6666 na tabela “Endereco_Clientes”.

Sendo assim, a coluna “Endereço” ficou sem nenhuma informação.

Perceba que pouco importou o campo “end_cli.codigo_cliente” estar do lado direito ou esquerdo da comparação.

O que realmente importa é o símbolo (+) estar ao lado da tabela correta em que não há correspondência exata de chave.

O recurso de “Outer Join” deve ser usado com cuidado.

Imagine se nossa tabela de clientes, houvessem milhões de registros (linhas) e erroneamente, fizéssemos um “Outer Join” sem a intenção de retornar todos os registros da tabela de clientes.

Além do fato de a consulta demorar muito mais tempo para ser executada, já que vai retornar milhões de registros (linhas), isso causaria uma lentidão desnecessária em todo o banco de dados, além de consumir desnecessariamente recursos de memória para que fosse possível retornar todos os milhões de registros da consulta.

Agora vamos extrair detalhadamente todas as colunas das tabelas “Cliente” e “Endereco_Clientes”.

```
SELECT cli.codigo_cliente
      , cli.cod_filial
      , cli.nome_cliente
      , cli.cnpj
      , end_cli.endereço
      , end_cli.numero
      , end_cli.complemento
      , end_cli.estado
      , end_cli.cidade
      , end_cli.telefone
FROM clientes      cli
      ,endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente
```

Veja que em ambos os exemplos de extração, seja utilizando o condicional “WHERE” ou o “From Inner Join”, tivemos que unir duas tabelas para que o relacionamento entre cliente e endereço fizesse sentido.

Na extração dos dados da query acima, estamos selecionando as colunas “codigo_cliente”, “cod_filial”, “nome_cliente” e “cnpj” da tabela “Clientes” e as colunas “endereço”, “numero”, “complemento”, “estado”, “cidade” e “telefone” da tabela “Endereco_Clientes”.

Para extrairmos o nome do cliente e seu respectivo endereço, unimos as tabelas “Clientes” e “Endereco_Clientes” através do código 9999 contido na coluna “codigo_cliente”, que está gravado em ambas as tabelas.

Tecnicamente falando, estamos unindo as tabelas “Clientes” e “Endereco_Clientes” através de sua chave primária com sua respectiva chave estrangeira.

Resultado:

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	ENDEREÇO	NUMERO	COMPLEMENTO	ESTADO	CIDADE	TELEFONE
1	9999	10	CLIENTE FILIAL	98765432198765	AV PAULISTA	1		SP	SAO PAULO	1198765432
2	9999	10	CLIENTE FILIAL	98765432198765	AV SAO JOAO	10		SP	SAO PAULO	1198765432

Notou que na instrução SQL, antes dos nomes de todas as colunas da tabela “Clientes” existe um “cli” e antes do nome de todas as colunas da tabela “Endereco_Clientes”, existe um “end_cli” ?

```
cli.codigo_cliente
,cli.cod_filial
,cli.nome_cliente
,cli.cnpj
, end_cli.endereço
, end_cli.numero
, end_cli.complemento
, end_cli.estado
, end_cli.cidade
, end_cli.telefone
```

Esses termos que antecedem as colunas e colocados logo após o nome das tabelas “Clientes” e “Endereco_Clientes”, são chamados de “Alias”

Alias

“Alias” são apelidos usados após a declaração de colunas ou tabelas, para simplificar ou modificar seus verdadeiros nomes durante a execução da consulta.

Exemplo prático: Normalmente uma pessoa que se chama Gabriela, não tem como apelido o termo “Gaby” ?

Nas consultas SQL, a utilidade é a mesma.

O “Alias” só tem validade durante a execução da consulta, dentro de uma instrução SQL.

Ao usarmos o “Alias” em uma tabela, informamos ao banco de dados que o nome original da tabela ou coluna que está sendo usada na consulta, será modificado, sendo substituído por um apelido.

Agora note que na tabela “Clientes”, existe uma coluna chamada “codigo_cliente” e na tabela “Endereco_Clientes”, também existe a mesma coluna, com o mesmo nome.

Ao declarar que desejamos exibir a coluna “codigo_cliente”, por meio do uso da instrução “Select”, é necessário informarmos ao banco de dados, de qual tabela desejamos exibir a coluna “codigo_cliente”.

Mas se ambas colunas possuem o mesmo nome e a mesma informação, por que o banco de dados precisa saber de qual tabela será extraída a informação, já que em ambas as tabelas, a informação é a mesma?

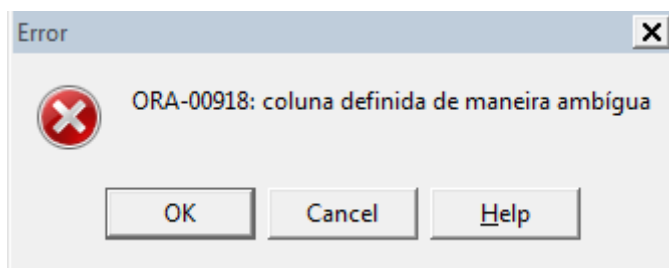
Para exemplificar, vamos recriar nossas tabelas “Clientes” e “Endereco_Clientes”, incluindo um novo campo chamado “STATUS” em ambas as tabelas.

Agora as duas tabelas, "Clientes" e "Endereco_Clientes", permanecem com a coluna "codigo_cliente", que continua sendo a mesma coluna chave para uni-las e também agora, temos uma nova coluna chamada "STATUS", que servirá para informar o status do cliente e status do endereço respectivamente, se estão ou não ativos, em ambas as tabelas.

Agora vamos consultar nossa nova coluna, chamada "STATUS".

```
SELECT status
FROM clientes cli
, endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente
```

Ao executar essa consulta no banco de dados, receberemos o erro abaixo:



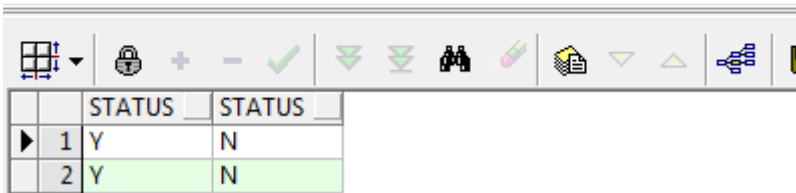
Perceba que o banco de dados exibiu esse erro, porque não consegue identificar de qual tabela desejamos exibir a coluna "STATUS", pois ela existe nas duas tabelas!

Agora vamos exibir a coluna "STATUS" das duas tabelas usando o Alias "cli" e "end_cli" de suas respectivas tabelas:


```

SELECT cli.status
      , end_cli.status
FROM clientes      cli
      ,endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente

```



	STATUS	STATUS
1	Y	N
2	Y	N

Note que na consulta acima, que a coluna "STATUS" da tabela "Clientes" está com o valor "Y".

Já na tabela "Endereco_Clientes", a coluna "STATUS" está com o valor "N".

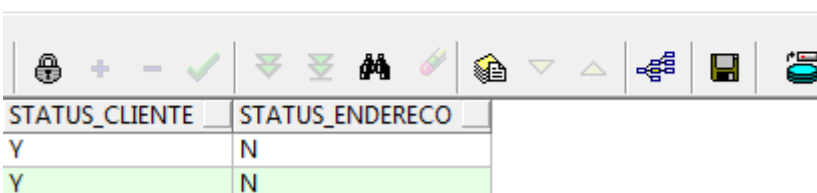
Mas e se desejássemos exportar o resultado dessa consulta para uma planilha, como saberemos de qual tabela se refere cada uma das colunas "STATUS", se elas possuem o mesmo nome?

Para resolver esse problema, podemos criar um "Alias" também para o nome da das colunas e assim, identificarmos a qual tabela se refere cada coluna de nome "STATUS".

```

SELECT cli.status      status_cliente
      , end_cli.status status_endereco
FROM clientes      cli
      ,endereco_clientes end_cli
WHERE cli.codigo_cliente = end_cli.codigo_cliente

```



STATUS_CLIENTE	STATUS_ENDERECO
Y	N
Y	N

Pronto !

Agora criamos os apelidos “status_cliente”, que se refere a tabela de clientes e o apelido chamado “status_endereço”, que se refere a tabela de endereços.

Com isso, saberemos exatamente de qual tabela representa cada coluna chamada de “STATUS”.

Você pode dar o nome de “Alias” que desejar para identificar cada coisa de sua consulta, da maneira que julgar melhor.



Um Detalhe: Caso queira usar um “Alias” com espaços, letras minúsculas e maiúsculas, acentuação ou qualquer outro caractere especial, será necessário declarar o “Alias” com o uso de um intervalo de caracteres do tipo apóstrofo (“ ”).

```
SELECT cli.status      "Status Do Cliente"
       , end_cli.status "Status do Endereço"
FROM   clientes        cli
       ,endereco_clientes end_cli
WHERE  cli.codigo_cliente = end_cli.codigo_cliente
```

Status Do Cliente	Status do Endereço
Y	N
Y	N

Uso dos Comparadores

Comparadores “=”, “<>”, “>”, “<”, “<=”, “>=”

Como o próprio nome sugere, usamos comparadores quando desejamos realizar consultas no banco de dados, utilizando uma ou mais comparações com o respectivo valor contido na coluna.

Igualdade: Símbolo “ = ”

Usamos esse comparador para testar uma condição de igualdade.

No exemplo abaixo, comparamos a igualdade entre a letra “Y” e o conteúdo da coluna “STATUS”.

```
SELECT *  
  FROM clientes  
 WHERE status = 'Y'
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
3333	1243	DML COMERCIO	78765432198765	Y	03/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
5555		CLINT GLACIAL	68719871897878	Y	06/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Resultado: Verdadeiro, pois a letra “Y” contida na coluna Status, é igual ao caractere “Y”, informado por nós, como parâmetro da consulta.

Diferença: Símbolo “ <> ”

Usamos esse comparador para testar uma condição de diferença.

No exemplo abaixo, comparamos a diferença entre a letra “Y” e o conteúdo da coluna “STATUS”

```
SELECT *
FROM clientes
WHERE status <> 'Y'
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
8888		MICAELA INDUSTRIA ...	12345678912345	N	02/03/2018

Resultado: Verdadeiro, pois o caractere “Y” é diferente de “N”

Minoridade: Símbolo “<”

Usamos esse comparador para testar uma condição de minoridade.

No exemplo abaixo, solicitamos ao banco de dados que nos retornasse somente os registros (linhas) cujo código do cliente seja menor do que o valor 4000.

```
SELECT *
FROM clientes
WHERE codigo_cliente < 4000
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
3333	1243	DML COMERCIO ...	78765432198765	Y	03/03/2018

Resultado: A consulta retornou o registro (linha) do “codigo_cliente” igual a 3333, que é menor do que o valor 4000, informado como parâmetro por nós, na consulta.

Maioridade: Símbolo “>”

Usamos esse comparador para testar uma condição de maioridade.

No exemplo abaixo, solicitamos ao banco de dados que nos retornasse somente os registros (linhas) cujo código do cliente fosse maior que o valor 4000.

```
SELECT *  
  FROM clientes  
 WHERE codigo_cliente > 4000
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
8888		MICAELA INDUSTRIA	12345678912345	N	02/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
5555		CLINT GLACIAL	68719871897878	Y	06/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Resultado: Todos os registros onde a coluna “código_cliente” é maior do que o valor 4000, foram retornados pela consulta.

Maioridade ou Igualdade: Símbolo “>= ”

Usamos esse comparador para testar uma diferença de maioridade, incluindo a igualdade.

No exemplo abaixo, desejamos que a consulta nos retorne os registros da coluna “codigo_cliente” que sejam maiores ou iguais ao valor 4444, informado por nós, como parâmetro.

```
SELECT *
  FROM clientes
 WHERE codigo_cliente >= 4444
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
8888		MICAELA INDUSTRIA	12345678912345	N	02/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
5555		CLINT GLACIAL	68719871897878	Y	06/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Resultado: Todos os resultados do campo “código_cliente” são maiores ou iguais ao valor 4444, informado por nós, como parâmetro da consulta.

Minoridade ou Igualdade: Símbolo “<= ”

Usamos esse comparador para testar uma diferença de minoridade, incluindo a igualdade.

No exemplo abaixo, desejamos que a consulta nos retorne os registros da coluna “codigo_cliente” que sejam menores ou iguais ao valor 4444.

```
SELECT *
  FROM clientes
 WHERE codigo_cliente <= 4444
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
3333	1243	DML COMERCIO	78765432198765	Y	03/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

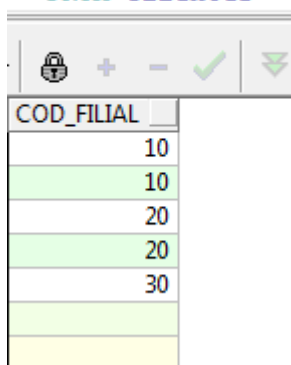
Resultado: Todos os resultados do campo “código_cliente” são menores ou iguais ao valor 4444, informado por nós, como parâmetro da consulta.

Função Distinct

Utilizamos a função *Distinct*, quando desejamos que a consulta nos retorne registros () únicos, sem repetições.

Primeiro, vamos executar um *Select* simples, retornando apenas a coluna "cod_filial" da tabela de clientes.

```
SELECT cod_filial  
FROM clientes
```



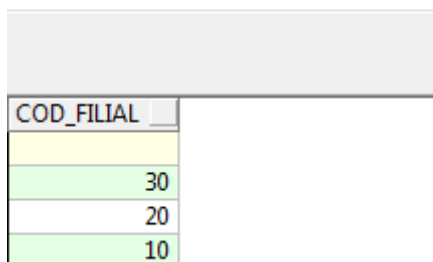
COD_FILIAL
10
10
20
20
30

Note que temos quatro registros com valores únicos na coluna "cod_filial".

Uma linha está nula e as demais apresentam os valores 10, 20 e 30 respectivamente.

Agora queremos que a consulta retorne todos os códigos únicos de filial da tabela *CLIENTES*, sem que apareçam resultados repetidos, usando o operador *DISTINCT*.

```
SELECT DISTINCT cod_filial  
FROM clientes
```



COD_FILIAL
30
20
10

Resultado: A consulta retornou quatro resultados únicos, sem repetições.

Operador Order By

É o operador SQL responsável por ordenar a exibição dos resultados durante a pesquisa.

Vamos adicionar a função “Order By” à pesquisa anterior.

Vamos então, ordenar a consulta através das colunas “cod_filial” e “nome_cliente”

```
SELECT DISTINCT cod_filial ,nome_cliente
  FROM clientes
 ORDER BY cod_filial,nome_cliente
```

COD_FILIAL	NOME_CLIENTE
10	MICAELA INDUSTRIA
10	MICHAEL JACKSON
20	DML COMERCIO
20	MICAELA FILIAL
30	INDUSTRIA SORRIDENTE
	CLINT GLACIAL
	COMERCIO DA ESQUINA

No exemplo acima, ordenamos os resultados primeiramente pela coluna “cod_filial” e depois pela coluna “nome_cliente”.

É possível também ordenarmos a consulta utilizando o número da coluna, ao invés do nome:


```
SELECT DISTINCT cod_filial ,nome_cliente,status
FROM clientes
ORDER BY 1,2
```

COD_FILIAL	NOME_CLIENTE	STATUS
10	MICAELA INDUSTRIA	N
10	MICHAEL JACKSON	Y
20	DML COMERCIO	Y
20	MICAELA FILIAL	Y
30	INDUSTRIA SORRIDENTE	Y
	CLINT GLACIAL	Y
	COMERCIO DA ESQUINA	Y

Veja que ao ordenamos a consulta pelo número de sua posição de exibição na tela, o número “1” se refere a coluna “cod_filial” e o número “2”, se refere a coluna “nome_cliente”.

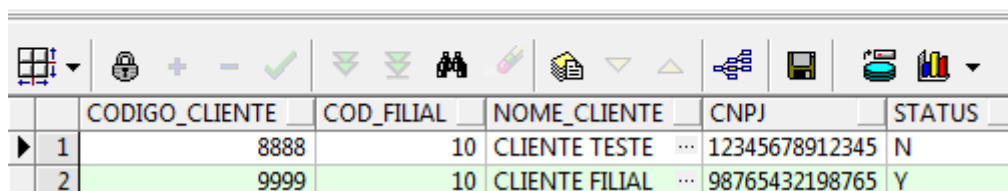
Independentemente do método que utilizarmos, o resultado será o mesmo.

Capítulo 9 – Operadores Condicionais

Operador IN e Not IN

A função do operador “IN” é retornar somente os registros pré-determinados que estiverem no intervalo da pesquisa.

```
SELECT *  
  FROM clientes  
 WHERE codigo_cliente IN (8888,9999)
```



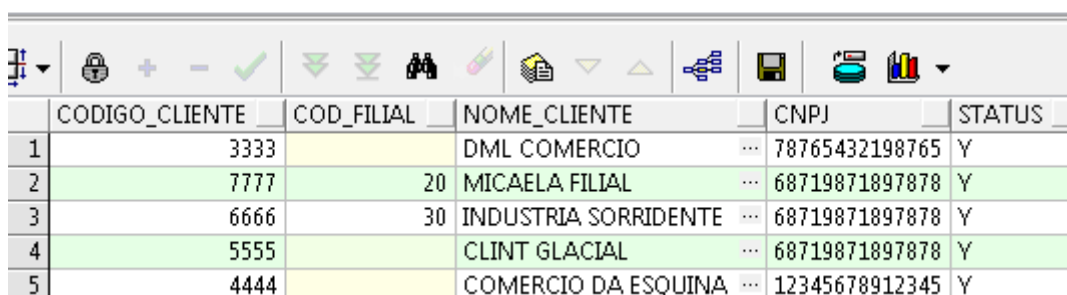
	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	8888	10	CLIENTE TESTE	12345678912345	N
2	9999	10	CLIENTE FILIAL	98765432198765	Y

Na consulta acima, desejamos que todas as colunas da tabela de clientes sejam exibidas (pelo uso do asterisco) ONDE o código do cliente esteja no intervalo 8888 e 9999 (somente esses dois).

Com o operador “NOT IN” ocorre justamente o inverso.

Os registros que estiverem definidos no intervalo não serão selecionados.

```
SELECT * FROM clientes  
 WHERE codigo_cliente  
 NOT IN (8888,9999)
```



	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	3333		DML COMERCIO	78765432198765	Y
2	7777	20	MICAELA FILIAL	68719871897878	Y
3	6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y
4	5555		CLINT GLACIAL	68719871897878	Y
5	4444		COMERCIO DA ESQUINA	12345678912345	Y

Operador Between

Usamos a função “Between” para pesquisar dados através de intervalos.

Vamos adicionar e popular uma nova coluna na tabela “Clientes” chamada de “Data_Criacao”.

```
SELECT * FROM clientes
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
1	9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
2	8888		MICAELA INDUSTRIA	12345678912345	N	02/03/2018
3	3333		DML COMERCIO	78765432198765	Y	03/03/2018
4	7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
5	6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
6	5555		CLINT GLACIAL	68719871897878	Y	06/03/2018
7	4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Agora vamos selecionar somente os registros em que a data de criação, esteja dentro do intervalo de 02/03/2018 e 05/03/2018.

```
SELECT * FROM clientes
WHERE data_criacao
BETWEEN '02/03/2018' AND '05/03/2018'
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
8888		MICAELA INDUSTRIA	12345678912345	N	02/03/2018
3333		DML COMERCIO	78765432198765	Y	03/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018

A função “Between” também pode ser utilizada com números.

```
SELECT * FROM clientes
WHERE CODIGO_CLIENTE
BETWEEN 3333 AND 6666
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
3333		DML COMERCIO ...	78765432198765	Y	03/03/2018
4444		COMERCIO DA ESQUINA ...	12345678912345	Y	07/03/2018
5555		CLINT GLACIAL ...	68719871897878	Y	06/03/2018
6666	30	INDUSTRIA SORRIDENTE ...	68719871897878	Y	05/03/2018

Operador Like

A palavra “Like” em inglês, é uma preposição que significa “Tal como”

A função “Like” na linguagem SQL, é representada pelo símbolo de porcentagem (%).

Utilizamos a função “Like”, quando desejamos pesquisar por um termo específico dentro de uma determinada coluna.

Podemos utilizá-lo de três formas:

1- Definimos um termo inicial e o símbolo “%” fica no final do termo pesquisado.

```
SELECT *
FROM clientes
WHERE nome_cliente LIKE 'MIC%'
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	MICHAEL JACKSON ...	98765432198765	Y
2	8888	10	MICAELA INDUSTRIA ...	12345678912345	N

No exemplo acima, desejamos procurar por todos os clientes que comecem pelo termo "MIC".

Resultado: A consulta retornou os registros "MICHAEL JACKSON" e "MICAELA INDUSTRIA"

2 - Definimos um termo final e o símbolo "%" fica no início do termo.

```
SELECT *  
  FROM clientes  
 WHERE nome_cliente LIKE '%ENTE'
```

		CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
▶	1	8888	10	TESTE CLIENTE	12345678912345	N
	2	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y

No exemplo acima, desejamos procurar por todos os clientes que tenham o termo "ENTE" no final.

Resultado: A consulta retornou os registros "TESTE CLIENTE" e "INDUSTRIA SORRIDENTE"

3- Definimos um termo intermediário, colocando o símbolo "%" no início e fim do termo.

```
SELECT *  
  FROM clientes  
 WHERE nome_cliente LIKE '%TRIA%'
```

		CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
▶	1	8888	10	MICAELA INDUSTRIA	12345678912345	N
	2	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y

No exemplo acima, desejamos procurar por todos os clientes que tenham o termo “TRIA” em qualquer lugar.

Resultado: A consulta retornou os registros “MICAELA INDUSTRIA” e “INDUSTRIA SORRIDENTE”

Veja que a localização do termo “TRIA” na coluna “nome_cliente”, não teve nenhuma importância.

Ao utilizarmos o termo “TRIA” entre dois sinais de porcentagem (% %), o termo poderia estar no início, no meio ou no final do texto, que a consulta seria retornada com sucesso.

Funções Count e Group By

Usamos a função “Count” quando queremos contar a quantidade de registros retornados pela pesquisa.

Vamos analisar a nossa tabela de clientes.

```
SELECT *  
FROM clientes
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	MICHAEL JACKSON	98765432198765	Y
2	8888		MICAELA INDUSTRIA	12345678912345	N
3	7777	20	MICAELA FILIAL	68719871897878	Y
4	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y
5	5555		CLINT GLACIAL	68719871897878	Y

Como temos poucos registros, sabemos de antemão que existem 5 clientes cadastrados, mas e se houvessem milhares ou milhões de registros?

Nesse caso, usariamos a função *COUNT* para contar todos os registros existentes na tabela.

```
SELECT COUNT(*)  
FROM clientes
```

	COUNT(*)
1	5

Agora vamos além de contar, agrupar os dados obtidos pela pesquisa.

```
SELECT COUNT(CNPJ) QTD_CNPJ  
      ,cnpj      NUM_CNPJ  
FROM clientes  
GROUP BY CNPJ
```

	QTD_CNPJ	NUM_CNPJ
1	1	98765432198765
2	2	12345678912345
3	2	68719871897878

No exemplo acima, queremos contar a quantidade de CNPJ cadastrado, agrupando aqueles que forem iguais.

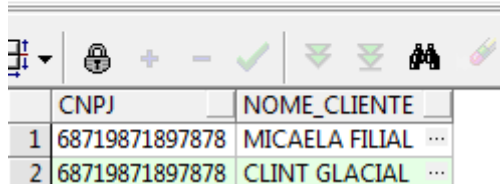
Temos então, 2 CNPJ iguais nas empresas “MICAELA INDUSTRIA” e “INDUSTRIA SORRIDENTE”.

```
SELECT cnpj  
      , nome_cliente  
FROM clientes  
WHERE cnpj = '12345678912345'
```

	CNPJ	NOME_CLIENTE
1	12345678912345	MICAELA INDUSTRIA ...
2	12345678912345	INDUSTRIA SORRIDENTE ...

Temos também 2 CNPJ iguais nas empresas “MICAELA FILIAL” e “CLINT GLACIAL”.

```
SELECT cnpj
      , nome_cliente
FROM clientes
WHERE cnpj = '68719871897878'
```



	CNPJ	NOME_CLIENTE
1	68719871897878	MICAELA FILIAL ...
2	68719871897878	CLINT GLACIAL ...



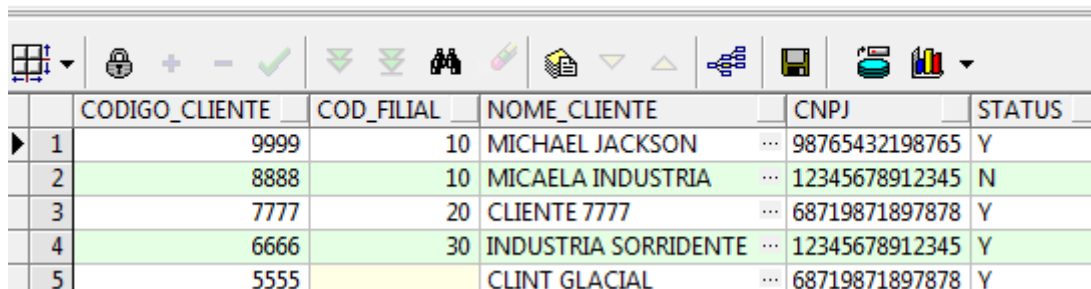
Note que esse é apenas um exemplo didático, pois como você já sabe, no mundo real, não existem duas empresas com o mesmo número de CNPJ.

Operadores Null e Not Null

A função “Null” é utilizada quando precisamos consultar registros em que uma determinada coluna ainda não tenha sido preenchida.

Primeiramente, vamos pesquisar novamente nossa tabela de clientes.

```
SELECT *
FROM clientes
```



	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	MICHAEL JACKSON	98765432198765	Y
2	8888	10	MICAELA INDUSTRIA	12345678912345	N
3	7777	20	CLIENTE 7777	68719871897878	Y
4	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y
5	5555		CLINT GLACIAL	68719871897878	Y

Vamos agora, procurar por registros em que a coluna “cod_filial” seja nula ou não preenchida.


```
SELECT *
  FROM clientes
 WHERE cod_filial IS NULL
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	5555		CLINT GLACIAL ...	68719871897878	Y

A função “Not Null” faz justamente o inverso da função “Null”, ou seja, procuramos por registros em que uma determinada coluna esteja obrigatoriamente com algum valor preenchido.

```
SELECT *
  FROM clientes
 WHERE cod_filial IS NOT NULL
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	MICHAEL JACKSON ...	98765432198765	Y
2	8888	10	MICAELA INDUSTRIA ...	12345678912345	N
3	7777	20	CLIENTE 7777 ...	68719871897878	Y
4	6666	30	INDUSTRIA SORRIDENTE ...	12345678912345	Y

Funções Exists e Not Exists

O condicional “Exists” é utilizado quando precisamos realizar uma checagem adicional, antes de retornar os registros (linhas) da consulta do “Select” principal.

O resultado do comando “Exists” ou “Not exists” é interno, ou seja, é validado dentro do banco de dados, se o retorno da validação é “verdadeiro” ou “falso”.

```

SELECT *
  FROM clientes
 WHERE codigo_cliente = 8888
    AND EXISTS ( SELECT 'VERDADEIRO'
                  FROM clientes
                  WHERE codigo_cliente = 7777
                )

```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
8888	10	MICAELA INDUSTRIA ...	12345678912345	N	02/03/2018

No exemplo acima, estamos pesquisando pelo cliente com código 8888, desde que algum cliente com o código 7777 **também exista**.

Note que se estivessemos pesquisando apenas pelo código de cliente que fosse igual a 8888, a consulta seria executada com sucesso:

```

SELECT *
  FROM clientes
 WHERE codigo_cliente = 8888

```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	8888		MICAELA INDUSTRIA ...	12345678912345	N

Porém a consulta principal precisa aguardar a checagem interna de validação contida dentro da instrução "Exists":

```

SELECT 'VERDADEIRO'
  FROM clientes
 WHERE CODIGO_CLIENTE = 7777

```

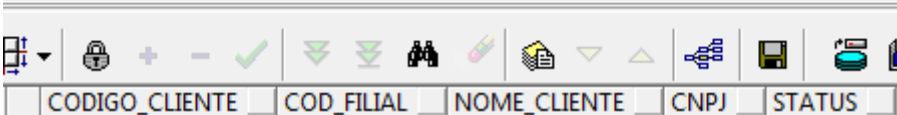
	'VERDADEIRO'
1	VERDADEIRO

Observação: O comando "Exists" valida apenas se a instrução SQL contida em seu interior, retorna algum registro.

Se a consulta retornar algum registro, então a condição de retorno será verdadeira e caso não retorne nenhum registro, a condição de retorno será falsa.

Com a condição "Not Exists" acontece justamente o oposto, ou seja, a condição dentro da instrução "Not Exists" não pode ser verdadeira.

```
SELECT *
FROM clientes
WHERE codigo_cliente = 8888
AND NOT EXISTS ( SELECT 'VERDADEIRO'
                  FROM clientes
                  WHERE CODIGO_CLIENTE = 7777 )
```



Note que com o uso do "Not Exists", as coisas se inverteram.

A pesquisa não retornou nenhum registro devido a condição interna dentro da instrução "Not Exists" ser verdadeira, ou seja, existe um cliente cadastrado dentro da tabela de clientes com o código 7777.

Outras Funções

Operador AND

O operador "And" é utilizado quando queremos incrementar o nível condicional da consulta em mais um nível, fornecendo mais um argumento.

Vamos estudar a pesquisa abaixo:

```
SELECT *
  FROM clientes
 WHERE cnpj = 12345678912345
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	8888		MICAELA INDUSTRIA ...	12345678912345	N
2	6666	30	INDUSTRIA SORRIDENTE ...	12345678912345	Y

Essa pesquisa retornou dois registros.

Agora vamos fornecer mais um argumento à pesquisa:

```
SELECT *
  FROM clientes
 WHERE cnpj = 12345678912345
       AND status = 'Y'
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	6666	30	INDUSTRIA SORRIDENTE ...	12345678912345	Y

Note que agora a consulta retornou apenas um registro (linha).

*Elaboramos a consulta para retornar apenas os registros onde o CNPJ seja igual a "12345678912345" **E** também o status seja igual a 'Y'.*

*A condição do CNPJ ser igual a "12345678912345" deve ser atendida, **E** a condição do status ser igual a "Y" também deve ser atendida, para que a consulta retorne um resultado.*

Se trouxéssemos esse exemplo para a vida cotidiana seria algo como:

*"Amanhã só vou trabalhar, se não houver greve **E** também se não estiver chovendo".*

Veja que se apenas não houver greve, não será o suficiente para que essa pessoa vá ao trabalho, da mesma forma que se apenas o tempo estiver bom, não será o suficiente para que ele vá ao trabalho.

Será necessário então, que as duas condições sejam atendidas.

Operador OR

De uma maneira simples, podemos dizer que o operador “OR” oferece opções à consulta caso alguma das condições propostas, não seja verdadeira.

Vamos estudar novamente a pesquisa abaixo:

```
SELECT *  
  FROM clientes  
 WHERE cnpj = 12345678912345
```



	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	8888		MICHAELA INDUSTRIA	12345678912345	N
2	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y

Como você já sabe, a pesquisa acima só será verdadeira caso o CNPJ do cliente seja igual a “12345678912345”.

Agora, vamos incluir na consulta, o operador condicional “OR”.

```
SELECT *
FROM clientes
WHERE cnpj = 12345678912345 OR codigo_cliente = 7777
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
2	7777	20	MICAELA FILIAL	68719871897878	Y
3	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y
1	8888		MICAELA INDUSTRIA	12345678912345	N

Agora, a consulta retornou um registro a mais, pois as duas condições foram atendidas.

Temos dois registros onde o CNPJ é igual a 12345678912345 e também um registro, onde o código do cliente é igual a 7777.

Outro exemplo:

```
SELECT *
FROM clientes
WHERE cod_filial = 465461465416
AND codigo_cliente = 168718671871
AND nome_cliente = 'AADSFASDFASDF'
OR STATUS = 'Y'
```

	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1	9999	10	MICHAEL JACKSON	98765432198765	Y
2	7777	20	MICAELA FILIAL	68719871897878	Y
3	6666	30	INDUSTRIA SORRIDENTE	12345678912345	Y
4	5555		CLINT GLACIAL	68719871897878	Y

Note que não existe em nossa tabela de clientes, nenhum cliente com código de filial igual a 465461465416, também não existe nenhum cliente com código igual a 168718671871 e muito menos um cliente chamado "AADSFASDFASDF".

Mas temos casos de clientes em que o status seja igual a "Y".

O fato de haver clientes com a coluna status igual a "Y" invalida toda a validação de igualdade que impomos na consulta, antes de usarmos o operador "OR".

Ao utilizamos o operador “OR”, basta que uma única condição após seu uso seja verdadeira, para que toda a consulta retorne como verdadeira no banco de dados.

Vamos tentar simplificar, fazendo um paralelo a uma situação do dia a dia.

Imaginemos que hoje, um suposto advogado esteja planejando sair para jantar, mas está em dúvida quanto ao lugar e ainda não definiu para onde pretende ir.

Esse advogado deseja sair para jantar em um restaurante que tenha obrigatoriamente as seguintes opções no cardápio: saladas, lasanha, salmon, picanha.

Porém, nosso advogado também aceita sair para jantar em outro tipo de lugar, desde que seja em uma pizzaria.

Perceba que se nosso suposto advogado sair para jantar em um restaurante que tenha todas as opções de comida do cardápio acima, já será o suficiente para encerrar o assunto, mas caso ele encontre uma única opção de pizzaria, também já atende as condições do nosso advogado.

Comer salada, lasanha, salmon e picanha é uma exigência, mas basta surgir uma opção de pizza, para que toda a exigência de comidas, perca o sentido.

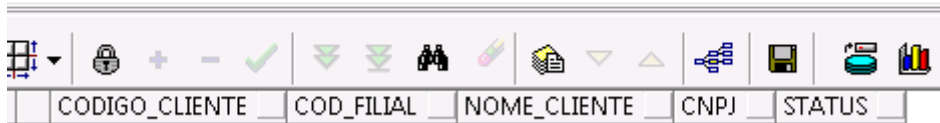
Assim, concluímos que qualquer uma das duas opções de lugar, restaurante ou pizzaria, atende a necessidade do nosso advogado, não necessitando que sejam as duas coisas juntas.

Agora, vamos utilizar os dois operadores, “And” e “Or” em uma mesma consulta:

```

SELECT *
  FROM clientes
 WHERE cod_filial      = 465461465416
       AND codigo_cliente = 168718671871
       AND (nome_cliente = 'AADSFA SDFASDF' OR STATUS = 'Y')

```



No exemplo acima, adicionamos à última linha, uma condição do tipo “And” e colocamos entre parênteses duas condições, que são a de o nome do cliente ser igual a “AADSFA SDFASDF” **OU** o status ser igual a letra “Y”.

No exemplo acima, o fato de nossa consulta existir um operador “OR” não invalidou todas as demais condições anteriores, pois essa condição do tipo “OR” ficou isolada entre dois parênteses, sob a “supervisão” de um operador do tipo “AND”.

Sendo assim, a condição de nossa consulta só será verdadeira se:

O código da filial for igual a “465461465416” **E** o código do cliente for igual a “168718671871” **E** (o nome do cliente for igual a “AADSFA SDFASDF” **OU** o status for igual a letra “Y”).

O fato de o conteúdo dentro dos parênteses, estarem sob a supervisão do operador “AND”, não fez com que toda a consulta fosse validada como verdadeira.

Perceba que não temos nenhum cliente com código de filial igual a “465461465416” **E** código de filial igual a “168718671871” **E** com nome igual a “AADSFA SDFASDF” **OU** seu respectivo status seja igual a letra ‘Y’.

Por esse motivo a consulta não retornou corretamente nenhum registro.

Função Having

Usamos a função de grupo “Having”, quando desejamos realizar uma consulta agrupando dados, desde que, satisfaçam a uma determinada condição.

Having Count

Vamos pesquisar por todos os clientes de nossa tabela de clientes

```
SELECT * FROM clientes for update
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
9999	10	MICHAEL JACKSON	98765432198765	Y
8888		MICAELA INDUSTRIA	12345678912345	N
7777	20	MICAELA FILIAL	68719871897878	Y
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y
5555		CLINT GLACIAL	68719871897878	Y
4444		COMERCIO DA ESQUINA	12345678912345	Y

Agora, vamos agrupar a consulta por CNPJ, porém, somente nos casos em que o resultado da soma do agrupamento de CNPJ, seja maior que 1.

```
SELECT COUNT(cnpj) cont_cnpj  
      , cnpj  
FROM clientes  
GROUP BY cnpj  
HAVING COUNT (cnpj) > 1
```

	CONT_CNBJ	CNPJ
1	2	12345678912345
2	3	68719871897878

Resultado: Temos 2 registros agrupados com CNPJ igual a “12345678912345” e 3 registros agrupados com CNPJ igual a “68719871897878”.

Having Sum

Agora, vamos utilizar a função de grupo “Having” junto com a função de soma “Sum”

Primeiramente, vamos criar uma nova tabela chamada “SALARIOS”.

```
SELECT * FROM salarios
```

COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
1	10	1000
2	10	5000
3	20	20000
3	20	2000
4	10	2000
5	20	6000

Agora com o uso da função “Having Count”, vamos selecionar somente os departamentos cuja soma dos salários seja maior que 8000.

```
SELECT COUNT(cod_departamento) count_dpto
      , cod_departamento      codigo_depto
      , SUM(salario)           soma_salarios
FROM salarios
GROUP BY cod_departamento
HAVING SUM(salario) > 8000
```

COUNT_DPTO	CODIGO_DEPARTAMENTO	SOMA_SALARIOS
3	20	28000

Note que o departamento de código 10 ficou de fora da pesquisa, devido ao fato de a soma de todos os salários do departamento, ser igual a 8000 e nós

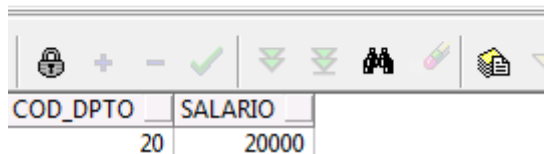
queremos selecionar somente os departamentos, cuja a soma dos salários seja superior a 8000.

Having Max

Vamos utilizar no exemplo abaixo, o operador "HAVING" juntamente com a função de grupo "MAX".

Nossa intenção é saber qual o maior salário dos departamentos, onde o salário seja maior do que 5000.

```
SELECT cod_departamento cod_dpto
      , MAX(salario)      salario
FROM salarios
GROUP BY cod_departamento
HAVING MAX(salario) > 5000
```



COD_DPTO	SALARIO
20	20000

Mas a consulta não deveria retornar todos os salários maiores que 5000?

A resposta é: Não.

Muitas pessoas fazem confusão ao utilizar as funções "Min" e "Max".

Ao utilizarmos a função "Max", informamos ao banco de dados que queremos saber, qual é o maior valor de uma coluna e não os registros que ultrapassem um determinado valor.

Caso seja necessário saber quais são os funcionários, cujo salário ultrapasse um determinado valor, devemos então, utilizar o condicional "Where" em conjunto com a função de grupo "Group By", como no exemplo abaixo:

```

SELECT COUNT(cod_departamento) qtd_dpto
      , cod_departamento      cod_dpto
  FROM salarios
 WHERE SALARIO > 4000
 GROUP BY cod_departamento

```

QTD_DPTO	COD_DPTO
2	20
1	10

Resultado: Temos 2 funcionários do departamento 20, que recebem salários superiores a 4000 e 1 funcionário do departamento 10, que recebe um salário superior a 4000.

Vamos agora fazer o inverso, utilizando a função “Min”.

Nossa intenção agora é saber qual é o menor salário por departamento, dos funcionários que ganham menos de 3000.

```

SELECT cod_departamento cod_dpto
      , MIN(salario)
  FROM salarios
 GROUP BY cod_departamento
 HAVING MIN(salario) < 3000

```

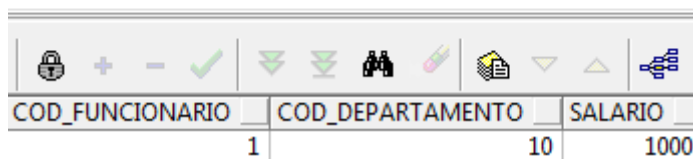
COD_DPTO	MIN(SALARIO)
20	2000
10	1000

Operador Union

A função *Union* é utilizada para consolidar o resultado de duas consultas.

Caso haja algum resultado repetido, não será retornado pela consulta.

```
SELECT *  
  FROM salarios  
 WHERE cod_funcionario = 1  
 UNION  
 SELECT *  
  FROM salarios  
 WHERE cod_funcionario = 1
```



The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons for query execution and editing. Below the toolbar, a table displays the results of the query. The table has three columns: COD_FUNCIONARIO, COD_DEPARTAMENTO, and SALARIO. The first row contains the values 1, 10, and 1000 respectively.

COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
1	10	1000

No exemplo acima, fizemos duas consultas rigorosamente iguais.

Apenas um registro foi retornado, já que a função “*Union*” não retorna registros que forem repetidos.



Quando usamos a função “*Union*” ou “*Union All*”, as consultas não necessariamente precisam ser de uma mesma tabela.

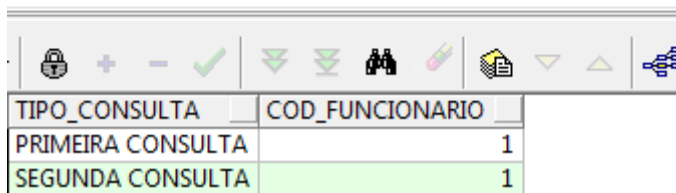
O importante é que ambas as consultas retornem a mesma quantidade de colunas e o mesmo tipo de dados.

Union All

A função “Union All” é similar a função “Union”.

A única diferença entre as duas funções, é a de que a função “Union All” retorna todos os resultados de ambas as consultas, independentemente de retornar resultados repetidos ou não.

```
SELECT 'PRIMEIRA CONSULTA' TIPO_CONSULTA
      , cod_funcionario
FROM salarios
WHERE cod_funcionario = 1
UNION
SELECT 'SEGUNDA CONSULTA' TIPO_CONSULTA
      , cod_funcionario
FROM salarios
WHERE cod_funcionario = 1
```



TIPO_CONSULTA	COD_FUNCIONARIO
PRIMEIRA CONSULTA	1
SEGUNDA CONSULTA	1



No exemplo acima, “inventamos” uma coluna chamada “TIPO_CONSULTA”.

Na primeira consulta, atribuímos o valor “PRIMEIRA_CONSULTA” e na segunda consulta, atribuímos o valor “SEGUNDA_CONSULTA” a coluna que criamos.

Depois unimos as duas consultas através da função “Union All”.

Note que essa coluna que “inventamos”, não existe em nenhuma tabela do banco de dados.

Nós a colocamos em nossa pesquisa, com a finalidade de identificar visualmente a qual consulta, o registro retornado se refere.

Você pode utilizar esse recurso em qualquer pesquisa e para qualquer finalidade que julgar necessário.

Note que tanto a primeira quanto a segunda consulta, são iguais, exceto a coluna que criamos.

Outro exemplo prático:

Vamos executar duas consultas rigorosamente iguais.

```
SELECT cod_funcionario
  FROM salarios
UNION ALL
SELECT cod_funcionario
  FROM salarios
```

COD_FUNCIONARIO	
	1
	1
	2
	2
	3
	3
	3
	4
	4
	5
	5

Resultado: O Código do funcionário se repetiu, pois o operador “UNION ALL” une duas ou mais consultas, não importando se os resultados retornados estão ou não repetidos.

Vamos refazer a consulta trocando a função “Union All” pela função “Union” ?

```
SELECT cod_funcionario
FROM salarios
UNION
SELECT cod_funcionario
FROM salarios
```

COD_FUNCIONARIO						
1						
2						
3						
4						
5						

Pronto. Agora temos todos os códigos dos funcionários, sem repetições.

Sub Query

Sub Query é um recurso muito útil da linguagem SQL.

O recurso de “Sub Query” permite que fontes de dados personalizados possam ser utilizada na pesquisa.

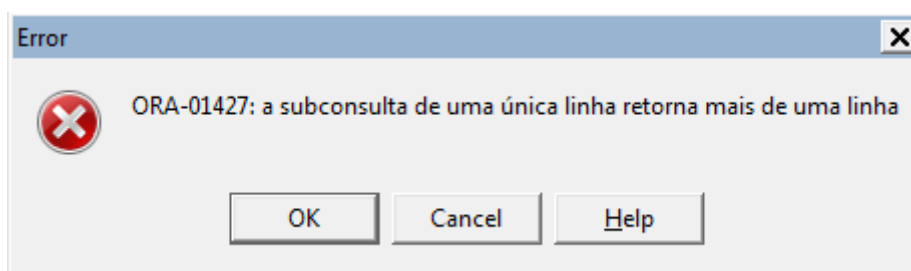
```
SELECT ( SELECT TELEFONE
          FROM ENDERECO_CLIENTES
          WHERE CODIGO_CLIENTE = CLI.CODIGO_CLIENTE
          AND ROWNUM = 1
        ) TELEFONE
      , CLI.*
FROM CLIENTES CLI
```

TELEFONE	CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
1139751423	9999	10	MICHAEL JACKSON	98765432198765	Y
1125431524	8888		MICAELA INDUSTRIA	12345678912345	N
	7777	20	MICAELA FILIAL	68719871897878	Y
	6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y
	5555		CLINT GLACIAL	68719871897878	Y
	4444		COMERCIO DA ESQUINA	12345678912345	Y

No exemplo acima, criamos uma coluna na consulta chamada “TELEFONE”, baseada em uma consulta externa.

Essa coluna baseada em uma consulta externa, necessariamente deve retornar um único registro e por isso, usamos por segurança a função ROWNUM e a igualamos a ao valor um, para que somente o primeiro registro da sub query seja retornado, eliminando a possibilidade de duplicidade.

Caso a coluna da consulta do tipo “Sub Query” retorne mais do que um registro, será disparado o erro abaixo pelo banco de dados:



Isso ocorreu intencionalmente, depois que nós propositalmente, duplicamos nossa tabela de endereços, repetindo o código 9999.

SELECT * FROM ENDereco_CLIENTES

CODIGO_CLIENTE	COD_FILIAL	ENDEREÇO	NUMERO	COMPLEMENTO	ESTADO	CIDADE	TELEFONE	STATUS
9999	10	AV PAULISTA	1		SP	SAO PAULO	1139751423	N
9999	20	AV SAO JOAO	10		SP	SAO PAULO	1125431524	N

Esse é um cenário que acontece frequentemente no “mundo real”, afinal o mesmo cliente pode ter mais do que um endereço.

É por esse motivo, que devemos ter cuidado ao utilizar em consultas, colunas baseadas em sub queries, pois basta uma simples alteração nos dados das tabelas, que são base para a sub consulta, para que a consulta inteira apresente erro.

Sub Query Utilizada Como Fonte De Dados

Outro recurso muito útil à nossas pesquisas em banco de dados, é utilizar sub queries, como fonte de dados.

Como você já sabe, a fonte de dados de todas as nossas consultas até agora, foram baseadas em tabelas, porém, podemos criar uma fonte de dados customizada, baseada em uma consulta.

```
SELECT sub_consulta.*
  FROM (SELECT cli.codigo_cliente
            , cli.nome_cliente
            , end_cli.endereço
            , end_cli.telefone
        FROM clientes      cli
            , endereco_clientes end_cli
        WHERE cli.codigo_cliente = end_cli.codigo_cliente
        ) sub_consulta
 WHERE sub_consulta.codigo_cliente IN (8888,9999)
```

CODIGO_CLIENTE	NOME_CLIENTE	ENDEREÇO	TELEFONE
9999	MICHAEL JACKSON	AV PAULISTA	1139751423
8888	MICAELA INDUSTRIA	AV SAO JOAO	1125431524

Veja que após a instrução “From”, não utilizamos uma tabela e sim uma “Sub Query” e atribuímos a ela um “Alias” chamado “Sub Consulta”.

Nossa “Sub Query” é composta dos campos “codigo_cliente”, “nome_cliente” da tabela “Clientes” e “endereço”, “telefone” da tabela “Endereco_Clientes”.

Podemos unir a Sub Query a uma outra tabela e até mesmo uni-la a uma outra Sub Query.

Nossa Sub Query gerou uma fonte de dados de 3 registros à nossa consulta principal:

CODIGO_CLIENTE	NOME_CLIENTE	ENDEREÇO	TELEFONE
9999	MICHAEL JACKSON	AV PAULISTA	1139751423
8888	MICAELA INDUSTRIA	AV SAO JOAO	1125431524
7777	MICAELA FILIAL	AV RIO GUARULHOS	1145432154

Na consulta principal, usamos o condicional WHERE normalmente e filtramos os resultados da sub consulta, retornando apenas os registros cujos clientes fossem igual a 8888 e 9999.

Capítulo 10 – Prática em Comandos de Manipulação de Dados

Data Manipulation Language

Como vimos nos primeiros capítulos, DML é uma linguagem de manipulação de dados dentro do SQL.

Para executar comandos de manipulação de dados (DML), seu usuário precisa ter permissão para manipular dados, caso a tabela a ser manipulada pertença a um outro usuário (Schema).

Os comandos de manipulação de dados, são: *Insert*, *Update* e *Delete*.

Após a execução de uma instrução do tipo DML, devemos efetivá-la ou descartá-la, através da instrução “Commit” ou “Rollback”.

Commit: É a instrução que efetiva a operação da manipulação de dados dentro do banco de dados.

Rollback: É a instrução que desfaz a operação da manipulação de dados, desde que não tenha sido efetivada anteriormente, com o uso do “Commit”.

Insert

INSERT é o comando DML responsável por incluir registros em uma tabela do banco de dados.

Sua sintaxe de construção do comando, é:

INSERT INTO [tabela] (coluna1, coluna2, coluna3...) VALUES ([valores a serem inseridos]).

Usando essa sintaxe para inserção de registros, precisamos informar apenas as colunas declaradas na instrução.

```
INSERT INTO clientes (codigo_cliente, nome_cliente, cnpj, status)
VALUES (3333, 'DML COMERCIO', 78765432198765, 'Y' );
COMMIT;
```

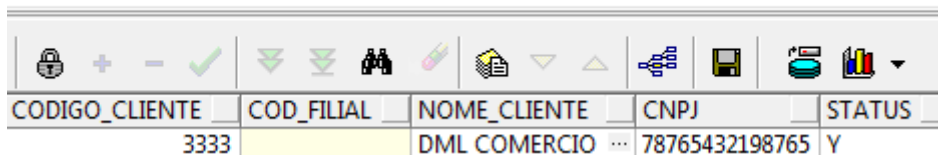
Insert: É a instrução DML que sinaliza ao banco de dados que estamos promovendo uma inclusão.

Into: Sinaliza em qual tabela será feita a inclusão.

Values: São os valores a serem inseridos em cada coluna da tabela.

Vamos verificar se nossa instrução DML de “Insert” funcionou.

```
SELECT * FROM clientes WHERE codigo_cliente = 3333
```



The screenshot shows a database query result in a table format. The table has five columns: CODIGO_CLIENTE, COD_FILIAL, NOME_CLIENTE, CNPJ, and STATUS. The first row contains the values 3333, (empty), DML COMERCIO, 78765432198765, and Y. The table is displayed with a standard database interface toolbar above it.

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS
3333		DML COMERCIO	78765432198765	Y

O registro 3333 foi inserido com sucesso!

Agora, vamos inserir um registro (linha), usando a instrução abaixo:

INSERT INTO [tabela] VALUES ([valores a serem inseridos])

Usando essa sintaxe para inserção de registros, será obrigatório o preenchimento de todas as colunas da tabela, até mesmo a das colunas que não sejam obrigatórias.

```
INSERT INTO clientes
VALUES (2222,NULL,'DML COMERCIO', 78765432198765,'Y' );
COMMIT;
```

Veja que usando esse tipo de instrução, tivemos que informar todos os valores a serem inseridos, mesmo para a coluna que não era obrigatória, como a coluna “cod_filial”.

Na coluna “Cod_Filial”, não inserimos nenhum valor, mas como somos obrigados a informar um valor à todas as colunas da tabela, devido ao tipo de sintaxe, inserimos o valor “NULL“, que na verdade, não representa nenhum valor.

Vamos verificar se o registro foi inserido.

```
SELECT *
FROM clientes
WHERE CODIGO_CLIENTE = 2222
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS										
2222		DML COMERCIO ...	78765432198765	Y										

Resultado: O registro com o código do cliente igual a 2222 foi inserido com sucesso!

É possível também inserir registros diretamente através de um comando Select, programas PL/SQL, etc.

Update

Update é o comando DML responsável por atualizar um ou mais registros de uma tabela do banco de dados.

Sua sintaxe de construção do comando, é:

UPDATE [tabela] SET [coluna] = [valor] WHERE [condição];

```
UPDATE clientes
  SET cod_filial    = 1234
    , status        = 'N'
 WHERE codigo_cliente = 2222;
COMMIT;
```

Update: É a instrução DML que sinaliza ao banco de dados, que estamos promovendo uma alteração em uma tabela.

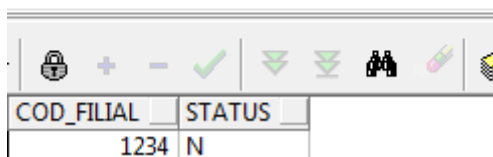
Set: É a instrução que define a alteração.

Em seguida informamos qual ou quais colunas serão alteradas e o seu respectivo valor.

Where: É a condição que será utilizada para selecionar quais registros devem sofrer a alteração.

Vamos verificar o resultado de nossa alteração:

```
SELECT cod_filial,status
  FROM clientes
 WHERE codigo_cliente = 2222
```



COD_FILIAL	STATUS
1234	N

Resultado: Os campos “Cod_Filial” e “Status” foram atualizados com os valores “1234” e “N” respectivamente, onde o código do cliente fosse igual a 2000.



Atenção: Devemos ter o máximo de cuidado quando estamos trabalhando com a instrução do tipo “Update”.

Todos os registros a serem alterados, devem ser cuidadosamente analisados e escolhidos através do condicional “Where”, antes de se efetuar a efetivação da operação, com uso da instrução Commit.

Já vi inúmeros casos de profissionais que, na correria do dia a dia para corrigir problemas de dados no sistema, acabam esquecendo de colocar o condicional WHERE ou calculando mal os registros a serem alterados em suas ações de correção.

Esse tipo de erro acarreta na atualização de toda a tabela ou de registros indevidos, provocando efeitos colaterais nada desejáveis.

Imagine cometer esse tipo de erro em uma tabela transacional com milhões de registros?

Delete

Delete é o comando DML responsável por excluir um ou mais registros de uma tabela do banco de dados.

Sua sintaxe de construção do comando, é:

DELETE [tabela] WHERE [condição];


```
DELETE clientes
WHERE codigo_cliente = 2222;
COMMIT;
```

COD_FILIAL	STATUS
1234	N

Delete: É a instrução DML que sinaliza que estamos promovendo uma exclusão de registros (linhas) no banco de dados. Em seguida informamos de qual tabela o registro será deletado.

Where: É a condição que será utilizada para selecionar quais registros devem ser deletados.

Vamos verificar o resultado de nossa deleção:

```
SELECT cod_filial,status
FROM clientes
WHERE codigo_cliente = 2222
```

COD_FILIAL	STATUS

Resultado: O registro com o código do cliente igual a 2222 foi excluído com sucesso!



Cuidado: Aqui vale a mesma recomendação dada à instrução UPDATE.

Vale e muito a pena gastar alguns minutos a mais, analisando as condições dos registros que serão deletados com o uso da cláusula WHERE.

FOR UPDATE

Ao utilizarmos a instrução “For Update”, estamos sinalizando ao banco de dados que desejamos reservar o registro selecionado para o nosso uso.

Se algum outro sistema ou usuário tentar manipular esse registro, usando alguma instrução SQL de manipulação de dados, ele terá que aguardar até que o registro seja totalmente liberado por nós.

Existem dois usos comuns a instrução “For Update”.

Um deles é quando usada dentro de programas PL/SQL, justamente para garantir que não haverá manipulação do dado, enquanto o registro estiver sendo utilizando por nós.

O outro, é quando utilizamos uma ferramenta gráfica de interface com o banco de dados.

Ao usarmos a instrução “For Update” com o uso de uma ferramenta gráfica, as colunas da consulta poderão ser manipuladas diretamente em tela, como se estivessem em uma planilha, sem a necessidade de usar nenhuma instrução SQL. Legal, não é?

Exemplo:

```
SELECT * FROM clientes FOR UPDATE
```

CODIGO_CLIENTE	COD_FILIAL	NOME_CLIENTE	CNPJ	STATUS	DATA_CRIACAO
9999	10	MICHAEL JACKSON	98765432198765	Y	01/03/2018
8888	10	MICAELA INDUSTRIA	12345678912345	N	02/03/2018
3333	20	DML COMERCIO	78765432198765	Y	03/03/2018
7777	20	MICAELA FILIAL	68719871897878	Y	04/03/2018
6666	30	INDUSTRIA SORRIDENTE	68719871897878	Y	05/03/2018
5555	320549723405	CLINT GLACIAL	68719871897878	Y	06/03/2018
4444		COMERCIO DA ESQUINA	12345678912345	Y	07/03/2018

Para executar a instrução “For Update”, seu usuário precisa ter permissão para manipular os dados da respectiva tabela.

Truncate

A instrução “Truncate” é uma instrução similar a instrução “Delete”.

Porém, sua função é excluir **Todos** os registros existentes na tabela.

A instrução “Truncate” não é considerada por muitos, como uma instrução DML, já que seu objetivo não é manipular e sim, sempre excluir todos os registros da tabela.

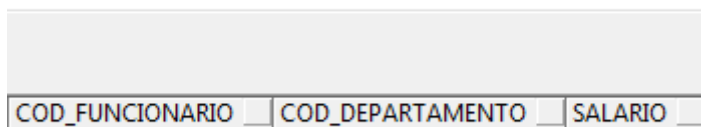
Sua sintaxe de construção do comando, é:

TRUNCATE table [nome da tabela]

```
TRUNCATE table salarios
```

Vamos verificar se nossa instrução de “Truncate” foi executada com sucesso.

```
SELECT * FROM salarios
```



COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
-----------------	------------------	---------

Resultado: Todos os registros da tabela “SALARIOS” foram excluídos.

Diferentemente da instrução “DELETE”, a instrução “TRUNCATE” não oferece a opção de escolher quais linhas se deseja excluir.

Observe que toda a tabela de salários foi deletada.

Qual a diferença entre a instrução “DELETE” quando utilizada para deletar todos os registros de uma tabela (sem utilizar a condição “WHERE”) e usar em seu lugar, a instrução “TRUNCATE”?

Resposta: Performance.

A instrução “TRUNCATE” é absurdamente mais rápida que a instrução “DELETE”, pois o banco de dados não precisa realizar nenhum tipo de checagem registro a registro antes de excluir os registros e nem reservar uma grande área para “Rollback, já que não há possibilidade de “voltar atrás” na exclusão dos registros excluídos.

Capítulo 11 – Prática em Funções De Conversão de Dados.

Funções são recursos que o banco de dados fornece para transformar dados de um formato para outro.

Há funções de data, conversão, analítica, numérica, texto e etc.

Funções De Data

Add_Months

A função “Add_Months” acrescenta um ou mais meses a uma data fornecida como parâmetro ou coluna de tabela do banco de dados.

Vamos usar a função “Add_Months” para adicionar 2 meses à data 1/1/2018.

```
SELECT add_months('1/1/2018',2) novo_mes  
FROM dual
```



	NOVO_MES
1	01/03/2018

Podemos também utilizar a função “Add_Months” de forma retroativa, usando um número negativo como parâmetro.

Vamos usar a função add_months para retroagir 2 meses à data 1/1/2018.

```
SELECT add_months('1/1/2018',-2) novo_mes
FROM dual
```

NOVO_MES	
1	01/11/2017

Extract

A função “Extract” retorna o valor numérico de uma data informada como parâmetro ou uma coluna do banco de dados do tipo data.

Vamos extrair o ano da data atual, passada como parâmetro.

```
SELECT EXTRACT (YEAR FROM SYSDATE) ANO FROM DUAL
```

ANO	
1	2018

Agora, vamos extrair o mês da data atual, passada como parâmetro.

```
SELECT EXTRACT (MONTH FROM SYSDATE) MES FROM DUAL
```

MES	
1	3

E por fim, vamos extrair o dia da data atual, passada como parâmetro.

```
SELECT EXTRACT (DAY FROM SYSDATE) DIA FROM DUAL
```

DIA	
1	8

Last Day

A função "LAST_DAY" retorna o último dia do mês de uma data informada ou de uma coluna de tabela do banco de dados, do tipo data

```
SELECT LAST_DAY('01/2/2018') ULTIMO_DIA_MES FROM DUAL
```

ULTIMO_DIA_MES	
1	28/02/2018

Months_Between

A função "Months_Between" calcula a diferença de meses dentro de um intervalo de datas.

```
SELECT MONTHS_BETWEEN ('01/12/2018','01/01/2018') CALCULA_MES
FROM dual;
```

CALCULA_MES	
	11

Next Day

A função “Next_Day” retorna o próximo dia do mês, utilizando um dia da semana como parâmetro.

O dia da semana é representado por uma tabela numérica, sendo:

1 – Domingo

2 – Segunda-Feira

3 – Terça-Feira

4 – Quarta-Feira

5 – Quinta-Feira

6 – Sexta-Feira

7 – Sábado

No exemplo abaixo, queremos saber qual será o dia do mês do próximo domingo, passando como parâmetro a data 08/03/2018.

```
SELECT NEXT_DAY('08/03/2018',1) next_day  
FROM dual;
```



NEXT_DAY
11/03/2018

Round

Usamos a função “Round”, quando precisamos arredondar um valor numérico, de acordo com a quantidade de casas decimais que fornecermos como parâmetro.

Logo após informarmos o valor numérico que desejamos arredondar, informamos a respectiva quantidade de casas decimais.


```
SELECT ROUND(123.987543,2) valor
FROM DUAL
```

VALOR
123,99

Resultado: O número 123.987543 foi convertido para o número 123,99, já que utilizamos 2 casas decimais como parâmetro de conversão.

Trunc

Uma das principais utilidades da função “Trunc”, é formatar uma coluna do tipo data (que por padrão retorna dia, mês, ano e hora) retornando apenas o formato dia, mês e ano.

```
SELECT TRUNC(SYSDATE) DATA
FROM DUAL
```

DATA
1 08/03/2018

Funções De Texto

Utilizamos as funções de texto quando precisamos formatar ou transformar textos ou colunas de uma tabela.

ASCII

A função “ASCII” retorna o número decimal correspondente da tabela “ASCII”, referente ao caractere informado como parâmetro.


A tabela “ASCII” é composta pela tabela abaixo :

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		Del

asciicharstable.com

Vamos aplicar no exemplo abaixo, o retorno do valor decimal referente ao caractere “%” (porcentagem), da tabela ASCII.

```
SELECT ASCII('%') porcentagem
FROM DUAL
```

	
PORCENTAGEM	
	37


Resultado: O valor decimal do caractere % (porcentagem) é 37.

Concat

A função “Concat” une duas sequências de caracteres ou campos do tipo texto.

Vamos fazer uma pesquisa em nossa tabela de clientes.

```
SELECT cod_filial
, nome_cliente
FROM clientes
WHERE codigo_cliente =3333
```

	
COD_FILIAL	NOME_CLIENTE
1243	DML COMERCIO ...

Agora vamos concatenar (juntar) uma descrição à coluna filial, com o objetivo de identificarmos a filial com um texto criado por nós.

```
SELECT concat(cod_filial,' - FILIAL SAO PAULO')
, nome_cliente
FROM clientes
WHERE codigo_cliente =3333
```

CONCAT(COD_FILIAL,'-FILIALSAOP	NOME_CLIENTE
1243 - FILIAL SAO PAULO ...	DML COMERCIO ...

Unimos o conteúdo da coluna “Cod_Filial”, ao texto “FILIAL SAO PAULO”

Uma outra forma de obter o mesmo resultado, é fazer a mesma junção usando o caracter “ || ” (pipe), por duas vezes.

O duplo caractere “pipe”, sinaliza ao banco de dados que o valor a ser retornado pela consulta, será concatenado com alguma outra coluna ou texto.

O valor concatenado pode ser um texto, um número, uma data ou uma outra coluna qualquer de uma tabela do banco de dados.

```
SELECT cod_filial || ' - FILIAL SAO PAULO'
       , nome_cliente
FROM clientes
WHERE codigo_cliente =3333
```

COD_FILIAL '-FILIALSAOPAULO'		NOME_CLIENTE
1243 - FILIAL SAO PAULO	...	DML COMERCIO ...



Note que a função CONCAT fez a junção do valor contido na coluna “Cod_Filial”, com o texto “FILIAL SAO PAULO”, porém, essa junção aparece apenas em nossa consulta.

O valor que ficou gravado na tabela “Clientes” do banco de dados, permanece o mesmo.

```
SELECT cod_filial
       , nome_cliente
FROM clientes
WHERE codigo_cliente =3333
```

COD_FILIAL	NOME_CLIENTE
1243	DML COMERCIO ...

Initcap

A função “Initcap” é utilizada para substituir a letra inicial de cada palavra, por uma letra maiúscula, mantendo as demais letras, em letras minúsculas.

```
SELECT INITCAP(nome_cliente) cliente
FROM clientes
WHERE codigo_cliente =3333
```

CLIENTE
Dml Comercio ...

Instr

A função “Instr” retorna a posição de um texto fornecido como parâmetro, dentro de uma determinada coluna ou texto.

Por padrão, a pesquisa é feita iniciando-se pela primeira letra no texto, à esquerda.

```
SELECT INSTR (nome_cliente, 'COMERCIO') cliente
FROM clientes
WHERE codigo_cliente =3333
```

CLIENTE
5

Resultado: O texto “COMERCIO” foi encontrado na posição 5 da coluna “Nome_Cliente da tabela “Clientes”, onde o código do cliente é igual a 3333.

NOME_CLIENTE
DML COMERCIO ...

Agora vamos procurar pela segunda ocorrência da letra “M”, na mesma pesquisa.

No parâmetro que está igual a 1, informamos qual será a posição inicial em que faremos a busca.

No parâmetro que está igual a 2, estamos informando qual é o número da ocorrência que estamos procurando.

Nesse caso estamos procurando pela segunda letra “M” do nome “DML COMERCIO”.

```
SELECT INSTR (nome_cliente, 'M', 1, 2) cliente
FROM clientes
WHERE codigo_cliente = 3333
```

CLIENTE	
7	

Length

A função “Length” é utilizada para contar a quantidade de caracteres de uma coluna de tabela do banco de dados ou de um texto informado como parâmetro.

```
SELECT LENGTH ('ESSE CURSO DE SQL É MUITO BOM') QTD_CARACTERES
FROM DUAL
```

QTD_CARACTERES	
29	

Lower

A função “Lower” é usada para colocar toda a sequência do texto em letras minúsculas.

```
SELECT LOWER('ESSE CURSO DE SQL É MUITO BOM') MINUSCULO  
FROM DUAL
```



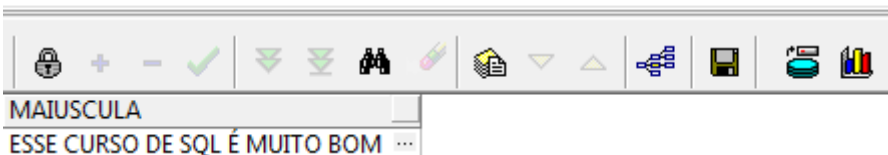
A screenshot of a SQL query result. The query is 'SELECT LOWER('ESSE CURSO DE SQL É MUITO BOM') MINUSCULO FROM DUAL'. The result is displayed in a table with one column named 'MINUSCULO' and one row containing the value 'esse curso de sql é muito bom'.

MINUSCULO
esse curso de sql é muito bom ...

Upper

A função “Upper” é usada para colocar toda a sequência do texto em letras maiúsculas.

```
SELECT UPPER('esse curso de sql é muito bom') MAIUSCULA  
FROM DUAL
```



A screenshot of a SQL query result. The query is 'SELECT UPPER('esse curso de sql é muito bom') MAIUSCULA FROM DUAL'. The result is displayed in a table with one column named 'MAIUSCULA' and one row containing the value 'ESSE CURSO DE SQL É MUITO BOM'.

MAIUSCULA
ESSE CURSO DE SQL É MUITO BOM ...

Lpad e Rpad

A função “Lpad” completa um texto ou a coluna de uma tabela, com um Algarismo informado como parâmetro, à esquerda.

Vamos analisar o caso abaixo:

COD_FILIAL	1243
------------	------

```
SELECT LPAD(COD_FILIAL,8,0) RESULTADO
FROM clientes
WHERE codigo cliente =3333
```

RESULTADO

00001243

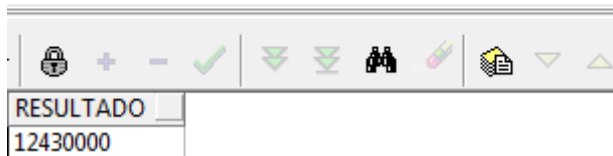
Rpad

Vamos utilizar o mesmo exemplo anterior.

COD_FILIAL	1243
------------	------

Vamos agora formatar a coluna “Cod_Filial” com 8 posições, completando os espaços em branco com o algarismo zero.

```
SELECT LPAD(COD_FILIAL,8,0) RESULTADO
FROM clientes
WHERE codigo_cliente =3333
```



The screenshot shows a database query result. At the top, there is a toolbar with various icons. Below it, a table with one column named 'RESULTADO' is displayed. The table contains a single row with the value '12430000'.

RESULTADO
12430000

Ltrim

A função “Ltrim” é usada para eliminar espaços em branco que estiverem à esquerda de um texto ou coluna de tabela do banco de dados.

```
SELECT LTRIM('        ESSE CURSO DE SQL É MUITO BOM        ') RESULTADO
FROM DUAL
```



The screenshot shows a database query result. At the top, there is a toolbar with various icons. Below it, a table with one column named 'RESULTADO' is displayed. The table contains a single row with the value 'ESSE CURSO DE SQL É MUITO BOM'.

RESULTADO
ESSE CURSO DE SQL É MUITO BOM

Perceba que apenas os espaços a esquerda foram eliminados. Os espaços que estão à direita permaneceram como estão.

Rtrim

A função “Rtrim” é usada para eliminar espaços em branco que estiverem à direita de um texto ou de uma coluna de tabela do banco de dados.

SELECT RTRIM(' ESSE CURSO DE SQL É MUITO BOM ') RESULTADO
FROM DUAL
RESULTADO
ESSE CURSO DE SQL É MUITO BOM ...

Resultado: Apenas os espaços à direita foram eliminados.

Trim

A função “Trim” é usada eliminar todos os espaços em branco de um texto ou coluna de tabela do banco de dados.

SELECT TRIM(' ESSE CURSO DE SQL É MUITO BOM ') RESULTADO
FROM DUAL
RESULTADO
ESSE CURSO DE SQL É MUITO BOM ...

Resultado: Todos os espaços em branco do texto foram eliminados.

Substr

A função “Substr” é usada para selecionar um trecho de um texto ou coluna de tabela do banco de dados, a partir de uma posição inicial e final, fornecida como parâmetro.

1 - Posição inicial do texto

2 - Posição final do texto

Replace

No exemplo abaixo, substituímos a expressão “NAO QUEREMOS” por “VAMOS”, do texto “HOJE NAO QUEREMOS ESTUDAR SQL”

RESULTADO	
HOJE VAMOS ESTUDAR SOL	...

Translate

A função “Translate” é parecida com a função “Replace”, com a diferença de que podemos alterar o resultado de forma parcial de todo o texto.

No exemplo abaixo, criamos o texto “HOJ4 V@MOS 4\$TUD@R SQL”.

Através da função “Translate”, alteramos todos os números “4” pela letra “E”, todos os caracteres “\$” pela letra “S” e todos os caracteres “@” pela letra “A”.

```
SELECT TRANSLATE('HOJ4 V@MOS 4$TUD@R SQL', '4$@', 'ESA') RESULTADO  
FROM DUAL
```

RESULTADO
HOJE VAMOS ESTUDAR SQL ...

Funções Numéricas

As funções numéricas são instruções SQL que realizam operações aritméticas.

ABS

A função “ABS” retorna o número absoluto de um número informado como parâmetro ou coluna de tabela do banco de dados.

```
SELECT ABS (-38) RESULTADO
FROM DUAL
```

RESULTADO
38

Podemos realizar qualquer operação matemática com a função “ABS”.

```
SELECT ABS (25.7 * -1) RESULTADO
FROM DUAL
```

RESULTADO
25,7

AVG

A função AVG é utilizada para retornar o resultado de uma média aritmética.

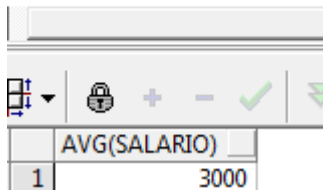
Vamos criar uma tabela chamada “SALARIOS”.

```
SELECT * FROM salarios
```

COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
1	10	1000
2	20	5000
3	30	3000

Agora vamos analisar, qual é a média de salário dos 3 funcionários dessa empresa.

```
SELECT AVG(salario)
FROM salarios
```



The screenshot shows a database query result window. At the top, there is a toolbar with icons for grid, lock, add, subtract, check, and refresh. Below the toolbar is a table with one column labeled 'AVG(SALARIO)' and one row with the value '3000'.

AVG(SALARIO)
3000

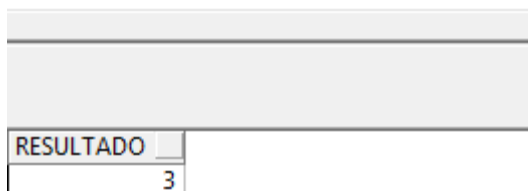
Resultado : $(1000 + 5000 + 3000) / 3 = 3000$

Count

A função "Count" é utilizada quando precisamos contar a quantidade de registros retornados pela consulta.

Vamos utilizar como exemplo, nossa tabela de salários, que tem 3 funcionários.

```
SELECT count(salario) resultado
FROM salarios
```



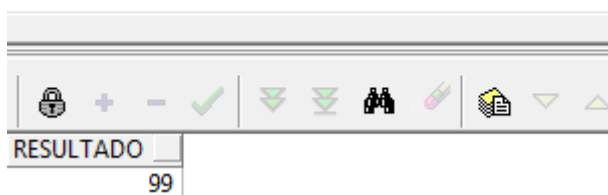
The screenshot shows a database query result window. At the top, there is a toolbar with icons for grid, lock, add, subtract, check, and refresh. Below the toolbar is a table with one column labeled 'RESULTADO' and one row with the value '3'.

RESULTADO
3

Greatest

A função “Greatest” retorna o maior de uma lista de expressões ou coluna de tabela do banco de dados.

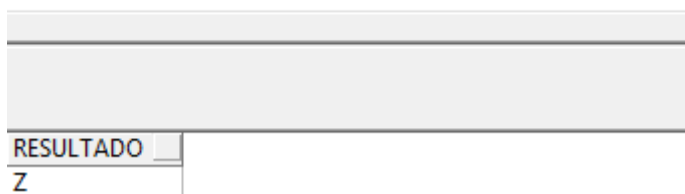
```
SELECT GREATEST (4,34,67,99) RESULTADO  
FROM DUAL
```



RESULTADO
99

Também podemos utilizar a função “Greatest”, para retornar o maior valor por ordem alfabética de um texto ou coluna do banco de dados.

```
SELECT GREATEST ('A','J','X','Z') RESULTADO  
FROM DUAL
```

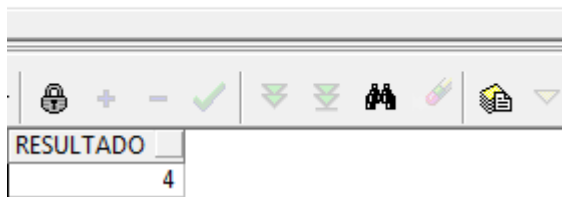


RESULTADO
Z

Least

A função “Least” faz o oposto da função “Greatest”, retornando o menor valor dentro de um intervalo de dados ou coluna de uma tabela do banco de dados.

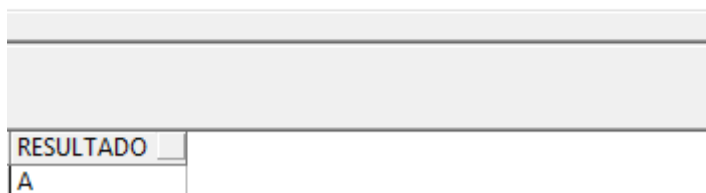
```
SELECT LEAST (4,34,67,99) RESULTADO  
FROM DUAL
```



RESULTADO
4

A função “Least” também pode ser utilizada com textos, buscando sempre, pelo menor valor.

```
SELECT LEAST ('K','M','G','A','Z') RESULTADO  
FROM DUAL
```



RESULTADO
A

Max

A função “Max” retorna o maior valor de uma lista de valores ou coluna de tabela.

Vamos novamente usar a tabela de salários como exemplo:


```
SELECT LPAD(COD_FILIAL,8,0) RESULTADO
FROM clientes
WHERE codigo_cliente =3333

SELECT * FROM salarios
```

COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
1	10	1000
2	10	5000
3	10	3000

Agora vamos retornar apenas o maior salário entre os 3 funcionários.

```
SELECT MAX(salario) resultado
FROM salarios
```

RESULTADO
5000

Agora, você pode estar se perguntando:

“Mas a função “Greatest” já não faz isso ?”

Imagine que além do maior salário, queremos também retornar o código do departamento.

Nesse caso, somente a função “Max” pode realizar essa tarefa, pois a função “Greatest” não é uma função de grupo.

Para obtermos o maior salário e o respectivo código do departamento:

```
SELECT cod_departamento
      , MAX(salario)      salario
FROM salarios a
GROUP BY cod_departamento
```

COD_DEPARTAMENTO	SALARIO
10	5000

Min

A função “Min” faz exatamente o oposto da função “Max”, retornando o menor valor de uma lista de valores ou coluna de tabela, com a mesma vantagem de ter a opção de agrupamento de dados.

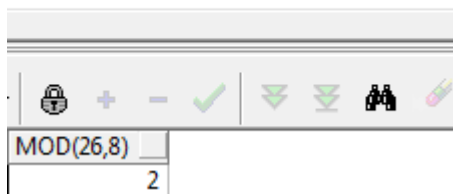
```
SELECT cod_departamento
      , MIN(salario)      salario
FROM salarios a
GROUP BY cod_departamento
```

COD_DEPARTAMENTO	SALARIO
10	1000

Mod

A função “Mod” retorna o resultado da diferença (resto) de uma divisão.

```
SELECT MOD(26,8) resultado
FROM dual
```



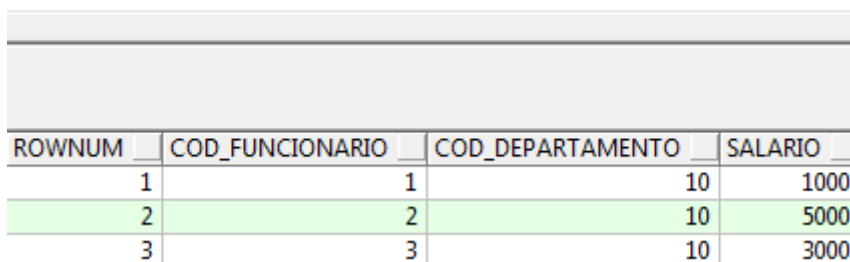
MOD(26,8)
2

Resultado 26 / 8 = 3, sobra 2

Rownum

Rownum é a função que retorna o número da linha, de acordo com o retorno da pesquisa feita pelo banco de dados.

```
SELECT ROWNUM, s.*
FROM salarios s
```



ROWNUM	COD_FUNCIONARIO	COD_DEPARTAMENTO	SALARIO
1	1	10	1000
2	2	10	5000
3	3	10	3000

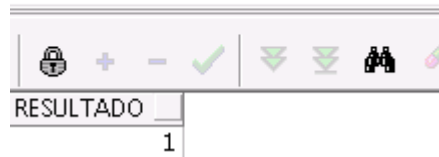
Sign

A função "Sign" retorna o sinal positivo ou negativo de uma operação aritmética.

Seu resultado sempre será 1, se o resultado da operação aritmética for um número positivo e -1, se o resultado retornado for um número negativo.

Abaixo uma operação matemática que resulta em um resultado positivo.

```
SELECT SIGN(1+1) resultado  
FROM DUAL
```



RESULTADO
1

Agora uma operação matemática que resulta em um resultado negativo.

```
SELECT SIGN ( (5 * 5) - 30) resultado  
FROM DUAL
```



RESULTADO
-1

Sum

A função “Sum” realiza somas entre valores de um intervalo de dados ou colunas de tabelas do banco de dados.

A função “Sum” também é uma função de grupo.

Vamos tornar a usar a tabela de salários como exemplo:

```
SELECT SUM(salario) soma_salarios
      ,cod_departamento
  FROM salarios
 GROUP BY cod_departamento
```

SOMA_SALARIOS	COD_DEPARTAMENTO
9000	10

Outras Funções

Case

A função “Case” nos permite personalizar o retorno de uma coluna de nossa consulta, com base em condições pré-definidas.

No exemplo abaixo criamos uma coluna chamada “nome_departamento” e o conteúdo será o retorno do resultado da função “Case”.

```

SELECT cod_funcionario,
CASE
    WHEN cod_departamento = 10
    THEN
        --
        'CONTABILIDADE'
        --
    WHEN cod_departamento = 20
    THEN
        --
        'RH'
        --
    ELSE
        --
        'OUTROS DEPARTAMENTOS'
        --
END nome_departamento
FROM salarios

```

COD_FUNCIONARIO	NOME_DEPARTAMENTO
1	CONTABILIDADE
2	CONTABILIDADE
3	CONTABILIDADE
4	RH
5	RH
6	OUTROS DEPARTAMENTOS

Note que utilizamos a seguinte lógica como retorno da função “Case”:

Caso o código do departamento seja igual a 10, então retornamos o valor “CONTABILIDADE”.

Caso o código do departamento seja igual a 20, então retornamos o valor “RH”.

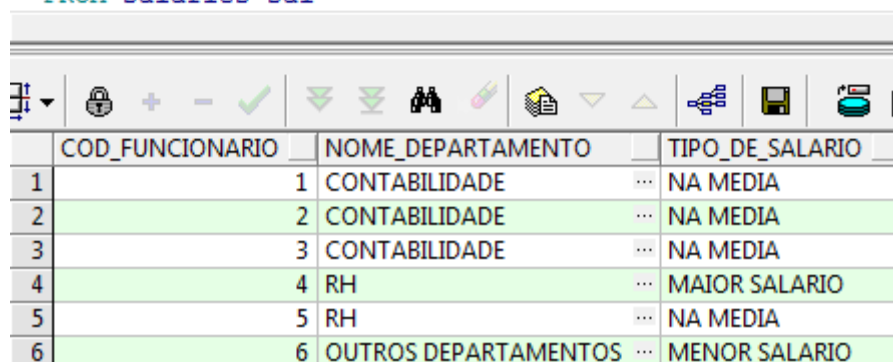
E em qualquer outro caso, retornamos o valor “OUTROS DEPARTAMENTOS”.

Nesse simples exemplo, estamos retornando um texto simples para a coluna “nome_departamento”, porém, é possível realizar também cálculos aritméticos, executar um outro Select, uma função, etc.

A função “Case” oferece inúmeras possibilidades.

Você pode utilizar a função “Case” em todas as colunas da pesquisa que julgar necessário, como em nosso outro exemplo abaixo:

```
SELECT SAL.COD_FUNCIONARIO,
--
CASE
    WHEN ( COD_DEPARTAMENTO = 10)
        THEN 'CONTABILIDADE'
    WHEN ( COD_DEPARTAMENTO = 20)
        THEN 'RH'
    ELSE
        'OUTROS DEPARTAMENTOS'
END NOME_DEPARTAMENTO,
--
CASE
    WHEN (SELECT
            MAX(SALARIO)
          FROM SALARIOS ) = sal.salario
        THEN
            'MAIOR SALARIO'
    WHEN (SELECT
            MIN(SALARIO)
          FROM SALARIOS ) = sal.salario
        THEN
            'MENOR SALARIO'
    ELSE
        'NA MEDIA'
END TIPO_DE_SALARIO
--
FROM salarios sal
```



	COD_FUNCIONARIO	NOME_DEPARTAMENTO	TIPO_DE_SALARIO
1	1	CONTABILIDADE	NA MEDIA
2	2	CONTABILIDADE	NA MEDIA
3	3	CONTABILIDADE	NA MEDIA
4	4	RH	MAIOR SALARIO
5	5	RH	NA MEDIA
6	6	OUTROS DEPARTAMENTOS	MENOR SALARIO

Veja que utilizamos a função “Case” para o retorno das duas colunas que criamos, “nome_departamento” e “tipo_salario”

A função “Case” é muito utilizada no mercado, devido a sua enorme flexibilidade.

Você pode resolver inúmeros problemas de lógica complexos.

Perceba que no exemplo anterior, não utilizamos sequer o condicional “Where” na elaboração de nossa consulta principal, mas você pode utilizá-lo à vontade, para combinar a pesquisa com outras tabelas, realizar filtros, agrupamentos e qualquer outra funcionalidade do SQL.



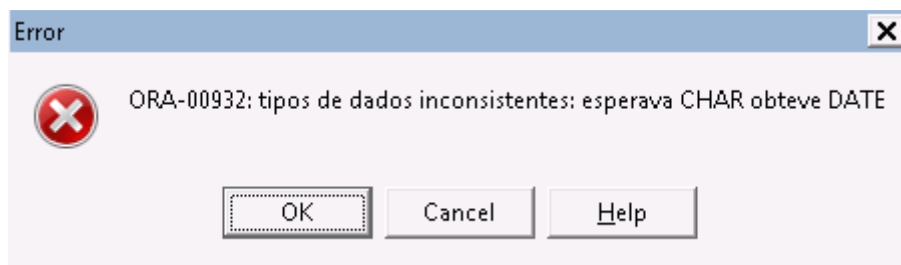
Vale lembrar que ao utilizarmos a função “Case”, o tipo de dado de retorno, deve ser sempre o mesmo.

Exemplo prático:

Você elaborou uma consulta utilizando a função “Case” em que na primeira condição, será retornado um texto.

Caso a primeira condição não seja atendida, você resolve então, retornar uma data.

Isso vai provocar um erro no banco de dados do tipo “tipo de dados inconsistentes”.



Caso uma das condições do “Case” seja verdadeira e desejarmos que seja retornado um texto, não podemos definir um retorno do tipo data, caso alguma outra condição seja atendida.

Para resolver esse tipo de problema, você deve converter todos os resultados para um formato texto, por exemplo, garantido um resultado sempre de mesmo tipo.

Decode

A função “Decode”, é um operador lógico de comparação.

Definimos o seu valor de retorno, de acordo com o resultado da comparação.

A sintaxe de construção da função “Decode”, é:

DECODE ([variável de comparação], [variável comparada], [valor de retorno], [retorno padrão])

Em nosso exemplo abaixo, nossa variável de comparação é a coluna “Cod_Departamento” da tabela “Salarios”.

Caso a variável de comparação seja igual a 10, retornamos o texto “RH”.

Caso a variável de comparação seja igual a 20, retornamos o texto “CONTABILIDADE”.

Caso a variável de comparação não seja igual a nenhuma das comparações, retornamos o texto “OUTROS DEPARTAMENTOS”.

```
SELECT DECODE(cod_departamento,10,'RH',20,'CONTABILIDADE','OUTROS DEPARTAMENTOS') NOME_DEPARTAMENTO
, cod_departamento
, salario
FROM salarios sal
```

	NOME_DEPARTAMENTO	COD_DEPARTAMENTO	SALARIO
1	RH	10	1000
2	RH	10	5000
3	RH	10	3000
4	CONTABILIDADE	20	6000
5	CONTABILIDADE	20	1500
6	OUTROS DEPARTAMENTOS	30	500

A função “Decode” é também muito utilizada no mercado, devido sua flexibilidade e fácil poder de compreensão, porém seu uso possui limitações.

Por exemplo, a função “Decode” só pode ser usada através de lógica por comparação.



O valor de retorno da função DECODE, deve ser também sempre do mesmo tipo de dado.

Exemplo: Se você definiu que o retorno de um “DECODE” será um texto para uma determinada condição, não pode permitir um retorno do tipo numérico, se uma outra condição for acionada.

Comandos DCL Na Prática

Vimos nos primeiros capítulos que DCL significa “Data Control Language” ou Linguagem de Controle de Dados, em português.

Sua função é dar privilégio a objetos do banco de dados criado por um outro usuário.

Exemplo prático real:

Todas as tabelas do Sistema de Gestão Empresarial da Oracle, o Oracle EBS, possui “owners” (usuários) específicos, separados por módulos.

O owner (usuário) GL é o “proprietário” das tabelas do módulo de Contabilidade, o General Ledger.

O owner (usuário) RA é o “proprietário” das tabelas do módulo de Contas a Receber, o Receivables.

O owner (usuário) OE é o “proprietário” das tabelas do módulo de Vendas, o Order Management.

O owner utilizado para execução de qualquer objeto no sistema Oracle Applications, é o APPS.

O usuário APPS precisa então de permissão (Grant) para acessar as tabelas e executar os programas dos módulos citados acima, que pertence aos “proprietários” GL, RA e OE.

Para conceder permissão a um usuário do banco de dados, usamos a seguinte sintaxe:

GRANT [tipo de privilégio] ON [objeto] TO [usuário];

```
GRANT SELECT ON salarios TO aplicacao ;
```

No exemplo acima, estamos concedendo privilégios de “Select” a tabela de salários a um usuário hipotético, chamado “aplicacao”.

As permissões mais utilizadas para tabelas, são:

SELECT, INSERT, UPDATE, DELETE, ALL

A opção de “Grant ALL” concede todos os privilégios ao usuário informado, à tabela desejada.

Caso desejássemos dar privilégio a todos os usuários do banco de dados, usamos a expressão “public” ao invés de informar um usuário específico.

Para conceder permissão de execução aos objetos do tipo “Procedure” ou “Package”, usamos o comando “Grant Execute”.

Para revogar uma permissão a um usuário do banco de dados, usamos a seguinte sintaxe:

REVOKE [privilégio] ON [objeto] FROM [usuário]

```
REVOKE SELECT ON salarios FROM aplicacao ;
```

No exemplo acima, estamos removendo a permissão de “Select” que havia sido dada ao usuário chamado “aplicacao”.

E com esse último comando, encerramos o nosso treinamento “SQL – O Guia Definitivo”.

Espero sinceramente ter correspondido a sua expectativa e contribuído de alguma forma para o seu desenvolvimento profissional.

Será um prazer saber em um futuro breve, que esse treinamento impactou de forma positiva em sua valorização profissional.

O tempo é o ativo mais valioso que os seres humanos possuem, pois ele representa literalmente a nossa vida, portanto, acredito que a “nossa vida” deve ser muito bem remunerada pelo mercado.

Isso só acontece, quando nos desenvolvemos pessoal e profissionalmente.

Na minha opinião, não há investimento melhor do que aquele que fazemos em nós mesmos.

Invista sempre em coisas que te levem ao próximo nível.

Lembre-se sempre: “Só o Conhecimento Traz o Poder”

“Só o conhecimento traz o poder.”

Sigmund Freud
fundador da psicanálise
(1856-1939)
O Pensamento Vivo de Freud

