

# **ESTRUTURA E BANCO DE DADOS**



Claudiney Sanches Junior

Unidade Curricular

# ESTRUTURA E BANCO DE DADOS



SÃO PAULO  
2018

FACULDADE DAS AMÉRICAS (FAM)

Diretora Geral: Leila Mejdalane Pereira

S19e

SANCHES JUNIOR, Claudiney.

Estrutura e banco de dados / Claudiney Sanches Junior. - São Paulo: SOCIEDADE EDUCACIONAL DAS AMÉRICAS, 2018.

408 p.

Núcleo de Educação a Distância - NEaD

Inclui bibliografia

ISBN 978-85-67379-31-9

1. Banco de dados - estrutura. I. Título.

CDU 004.6

PRODUÇÃO EDITORIAL - NÚCLEO DE EDUCAÇÃO A DISTÂNCIA - FACULDADE DAS AMÉRICAS

Direção Editorial: Denise Aparecida Campos Coordenação

Editorial: Vania Aparecida Marques Leite

Secretaria Editorial: Viviane Cavalcante de Sousa

Projeto Gráfico e Diagramação: Elis Nunes

Imagens: Arquivo pessoal

## ■ SOBRE O AUTOR

Mestre em Engenharia Elétrica pela Universidade de São Paulo (USP 2006) e Bacharel em Ciências da Computação pelo Centro Universitário Eurípides de Marília (2002). Tem experiência na área de Ciência da Computação, atuando principalmente nos seguintes temas: realidade virtual, modelagem 3D, lógica reconfigurável (fpga), gerenciamento e desenvolvimento de software em sistemas de tempo real, sistemas distribuídos, inteligência artificial, engenharia de software e computação gráfica. Professor na Faculdade das Américas. Atuou como docente na Universidade Cidade de São Paulo (UNICID), Universidade Cruzeiro do Sul (UNI-CSUL), Centro Universitário Adventista de São Paulo (UNASP) e como consultor de projetos na Universidade de São Paulo (LSI - USP/ LSI-Tec). Foi Coordenador do Laboratório Multimídia da Universidade de Santo Amaro (UNISA) e Diretor da Faculdade de Computação e Coordenador do Curso de Sistemas de Informação na modalidade presencial e a distância.

Endereço do currículo lattes:

<http://lattes.cnpq.br/7554696629208726>



# ■ SUMÁRIO

<b>AULA 1</b>	<b>Estrutura e Banco de Dados .....</b>	<b>11</b>
	Algoritmos, Estruturas de Dados e Tipos de Dados .....	12
	Estruturas de Dados Básicas.....	22
	Estruturas de Dados Avançado .....	37
<b>AULA 2</b>	<b>Introdução a Sistemas de Banco de Dados .....</b>	<b>45</b>
	A importância de estudar e aprender Banco de Dados .....	46
	Raízes históricas .....	48
	Problemas de gerenciamento de dados do sistema de arquivos.....	53
	Modelos de Dados, sua importância e diferenças .....	54
	Regras de Negócios e Evolução dos Modelos de Dados .....	55
<b>AULA 3</b>	<b>Modelagem de Dados.....</b>	<b>83</b>
	Etapas de um projeto .....	84
	Modelagem do E-R .....	86
	Atributos e chaves.....	89
	Cardinalidade.....	91
	Especialização ou Generalização .....	94
	Condisionalidade de Relacionamento .....	100
	Relacionamentos Contingentes.....	101
	Agregação .....	102
	Validação de Diagramas.....	102
	Aspecto Temporal.....	103
	Modelo Lógico.....	104
	Regras de integridade.....	108
<b>AULA 4</b>	<b>Ambiente de Trabalho .....</b>	<b>115</b>
	Introdução à Máquinas Virtuais .....	116
	Instalar um servidor Linux .....	125
	Instalar o SGBD Oracle .....	137
	Para instalar SQLDeveloper.....	141
<b>AULA 5</b>	<b>Normalização.....</b>	<b>143</b>
	Formas Normais .....	144
	Ferramentas Case.....	149
	Padronização no SGBD.....	150
	Introdução à Linguagem SQL.....	153

<b>AULA 6</b>	<b>Introdução ao SQL.....</b>	<b>175</b>
	Alterando Tabelas Existentes.....	176
	INSERT e conceitos iniciais da SELECT .....	181
	Consultas utilizando SELECT.....	183
	Expressões aritméticas e os operadores BETWEEN, IN, LIKE e IS NULL .....	190
	Operadores Especiais.....	191
	UPDATE e DELETE.....	198
<b>AULA 7</b>	<b>Manipulação de Dados com SQL .....</b>	<b>209</b>
	QUERY e SUBQUERY .....	210
	Agrupando dados .....	216
	Funções de data .....	224
<b>AULA 8</b>	<b>Funções em SQL .....</b>	<b>241</b>
	Introdução a Funções SQL.....	242
	Função para valores nulos - NVL.....	242
	Funções Alfanuméricas .....	244
<b>AULA 9</b>	<b>SQL Avançado.....</b>	<b>275</b>
	Junção de tabelas.....	276
	Junção de tabelas com ALIAS.....	285
	Operadores de junção de SQL .....	291
<b>AULA 10</b>	<b>SQL Procedural .....</b>	<b>317</b>
	Introdução ao PL/SQL .....	318
	Blocos no PL/SQL.....	318
	Declarações de Variáveis .....	322
	Declaração LOOP.....	325
	Procedure.....	338
	Cursor .....	342
	Sequence .....	347
	Function .....	350
	Triggers.....	352
	<b>Expectativa de Resposta.....</b>	<b>317</b>
	<b>Referências bibliográficas .....</b>	<b>407</b>

## ■ APRESENTAÇÃO

Bem vindo(a) aos estudos da Unidade Curricular de Estrutura e Banco de Dados.

Por meio dela apresentaremos à você as estruturas de dados, diferentes técnicas de implementação e os Bancos de Dados Relacionais. As estruturas de dados dão suporte para você desenvolver o raciocínio lógico. O estudo dos principais conceitos de estrutura de dados o ajudará a se tornar um programador melhor e a entender quais estruturas um Banco de Dados utiliza.

Abranger a importância do banco de dados e apresentar os principais termos da área fazem parte dos nossos objetivos. Você irá contemplar as raízes históricas dos bancos de dados e os problemas enfrentados com os sistemas de arquivos

Estudaremos também a modelagem de negócios de uma empresa, criando diagramas que vão lhe ajudar a compreender como a empresa funciona e suas necessidades de dados. Você vai entender as fases de projeto e modelagem de um Banco de Dados Relacional e aprender as fases envolvidas e os conceitos do modelo entidade-relacionamento.

O estudo de modelagem de banco de dados o ajudará a criar um conceito abstrato que poderá ser implementado em qualquer SGBD. Notaremos os desafios existentes em um projeto de Banco de Dados. Conhecer ferramentas para criar e administrar um SGBD lhe dará uma noção do tempo e do esforço necessário para instalar um servidor de banco de dados.

No desenvolvimento destes estudos você aprenderá instalar e preparar um Banco de Dados Relacional em um servidor Linux. Existem alguns padrões para Banco de Dados que podem lhe ajudar na modelagem, são as Formas Normais . Você deve entender o processo de normalização e quando criar um banco desnormalizado.

Apresentaremos à você uma Ferramenta Case que o ajudará a se adaptar rápido a diferentes empresas e você vai aprender a criar e alterar tabelas bem como inserir, alterar e excluir registros utilizando o SQL.

Enfim, você aprenderá diversos comandos utilizados na programação e desenvolverá conhecimentos e competências para ser um administrador de banco de dados (DBA) qualificado.

● Bom estudo!



# AULA 1

## ESTRUTURA E BANCO DE DADOS

### NESTA AULA

- » Algoritmos, Estruturas de Dados e Tipos de Dados
- » Estruturas de Dados Básicas
- » Estruturas de Dados Avançado

### ■ METAS DE COMPREENSÃO

- » Compreender algoritmos com foco em estruturas de dados.
- » Conhecer os tipos de dados.
- » Compreender o funcionamento de estruturas de dados básicas tais como: lista lineares, pilhas (FILO - First-In, Last-Out) e filas (FIFO - First-In, First-Out).
- » Implementar funções como Pop e Push.

### ■ APRESENTAÇÃO

Nosso objetivo nesta aula é apresentar o conteúdo que será estudado na unidade. Vamos focar nas estruturas de dados apresentando técnicas de implementações diferentes. As estruturas dão suporte para você desenvolver um raciocínio lógico para resolver um problema de processamento de dados. Sem esse raciocínio não tem sentido a programação e o armazenamento de dados.

O estudo dos principais conceitos de estrutura de dados o ajudará a se tornar um melhor programador e a entender quais estruturas um Banco de Dados utiliza e quando se tornará necessário utilizar um Banco de Dados.

# ■ ALGORITMOS, ESTRUTURAS DE DADOS E TIPOS DE DADOS

## Algoritmos

sequência de passos que visam atingir objetivos bem definidos em um determinado período de tempo.

Os **algoritmos** fazem parte do seu cotidiano. Você os encontra em instruções detalhadas, com uma sequência de passos objetivos e bem definidos. Um bom exemplo é a descrição passo a passo de como trocar a resistência de um chuveiro, ou montar um móvel ou cozinhar uma receita culinária.

Um algoritmo pode ser visto como uma sequência de passos que visa atingir objetivos bem definidos em um determinado período de tempo. É composto por instruções claras e bem definidas com o objetivo de resolver um dado problema. É um caminho que leva à solução, à uma norma de solução a ser trilhada. Sempre que executado, sobre as mesmas condições, produz o mesmo resultado.

## Linguagem de Programação

conjunto de instruções para traduzir o algoritmo para a linguagem do computador.



### pense nisso

Qual linguagem devo investir meu tempo e dinheiro? Quais fatores devo considerar?

Será que devo pesar o mercado atual ou devo prospectar o mercado futuro? Qual foi a linguagem mais utilizada no ano passado?

## Linguagem de Máquina

linguagem binária que somente o interpretador de comandos entende.

Você poderá escolher um vocabulário ou conjunto de instruções para traduzir o algoritmo para a linguagem que o computador entende. A **linguagem de máquina** é desconfortável para humanos e por isso foram desenvolvidas várias ferramentas de tradução que criam um conjunto de instrução que pode ser traduzido. Este vocabulário será a Linguagem de Programação. A escolha da Linguagem de Programação

pode depender de alguns fatores como: estrutura de dados, mercado, tempo, grau de dificuldade, orçamento do projeto, domínio de conhecimento e inclusive preferências pessoais.

Estrutura de **dados** e algoritmos estão ligados à medida que algumas **estruturas de dados** podem ser mais facilmente implementadas ou traduzidas em uma linguagem de programação do que em outra. Assim, é importante escolher como você vai representar os dados ou valores que serão utilizados para a resolução de problemas. A escolha da estrutura de dados irá depender das operações a serem realizadas sobre os dados.

Dados é definido como uma sequência de símbolos quantificáveis, por exemplo: um texto é um dado, sendo suas letras símbolos quantificados. Fotos, figuras, sons gravados e animação são dados mesmo se forem incompreensíveis para o usuário, ou seja, o usuário não comprehende os símbolos quantificados.

É importante diferenciar aqui dado de informação. Informação é uma abstração informal, isto é, não pode ser formalizada por meio de uma teoria lógica ou matemática, que está na mente de alguém, representando algo significativo para essa pessoa. Dados precisam de uma pessoa para produzir informação.

Os dados podem ser divididos em três categorias: simples, estruturados e abstratos.

### Dados

uma sequência  
de símbolos  
quantificáveis

### Estruturas de Dados

modo de representar  
os dados e organizar



Algoritmo - uma sequência de passos que visam atingir objetivos bem definidos em um determinado período de tempo.

Linguagem de Programação - conjunto de instruções para traduzir o algoritmo para a linguagem do computador.

Dados - uma sequência de símbolos quantificáveis.

## DADOS SIMPLES OU PRIMITIVOS

Os dados são armazenados temporariamente em variáveis de memória e para otimizar a utilização da memória, na maioria das Linguagens de Programação, são tipificados, criando espaços para números

inteiros, números reais, letras e **booleanos**. Os tipos de dados podem variar de acordo com a linguagem de programação escolhida, mas os tipos básicos de programação permanecem e são assim especificados:

**Inteiro:** pode armazenar números de qualquer natureza (nulo, positivo ou negativo) desde que pertença ao conjunto dos números inteiros. Em algumas linguagens como C, Java e JavaScript os inteiros são declarados com **int**. São exemplos de números inteiros 20; 0; -125 e 40.

**Real:** pode armazenar qualquer valor pertencente ao conjunto dos números reais e inteiros (nulo, positivo ou negativo). São exemplos de números inteiros -22,4; 0; 8 e 5,30. São representados como **float** ou **double** dependendo da precisão.

**Caractere:** pode armazenar valores alfanuméricos e caracteres especiais como por exemplo "Análise de Sistemas", "Rebecca", "Thaís" e "ok". São identificados na maioria das linguagens como **String** ao tratar de um conjunto, e **char** ao tratar de apenas uma letra ou literal.

**Lógico ou boolean:** pode armazenar apenas valores verdadeiro (V) – **true** ou falso (F) – **false**.



[saiba mais](#)

O site [w3schools.com](http://w3schools.com) apresenta um conteúdo fácil e didático de HTML, CSS e JavaScript oferecendo exercícios preparatórios para certificação.

O código abaixo apresenta como criamos estruturas de dados simples em JavaScript. O código pode ser gerado em qualquer **software** que permite a manipulação de texto, incluído o bloco de notas. A extensão do arquivo deve ser .html e deverá ser executado por um **browser** como Edge, Chrome ou Mozilla Firefox.



dica

Você pode utilizar diferentes ferramentas para lhe auxiliar como **Visual Studio Community** da Microsoft, **NetBeans**, **Brackets** entre outros. Teste todos os códigos abaixo e procure entender cada linha apresentada.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Variáveis em Javascript</title>
    <script>
      var Nome = "Claudiney Sanches";
      var Idade = 42;
      var Sexo = "M";
      var Trabalhando = true;
      var Salario = 5324.23;
    </script>
  </head>
  <body>
    </body>
  </html>

```

Observe que o JavaScript não necessita que informemos o tipo de dado que será alocado automaticamente. O JavaScript também torna opcional utilizar o comando var antes de declarar uma variável. Na linguagem Java, é necessário informar o tipo antes do nome da variável.



dica ||

Para testar os códigos apresentados em Java você poderá utilizar a ferramenta **NetBeans** com JDK. Acesse o link <https://netbeans.org/> para fazer o download da ferramenta. Recomendo que baixe a versão completa com suporte a Java SE, HTML/Javascript, C++, PHP e Java EE pois lhe ajudará em diferentes momentos no decorrer do curso. É importante observar que também será necessário instalar a última versão do JDK. Uma opção para facilitar sua instalação é baixar o **NetBeans** com o JDK em um único arquivo de instalação.

```

package variaveisJava;
public class variaveisJava {
  public static void main(String[] args) {

```

```

String Nome = "Claudiney Sanches";
int Idade = 42;
char Sexo = 'M';
boolean Trabalhando = true;
float Salario = 6543.21f;
}
}

```

Observe a diferença entre os códigos, pois, ambos definem as mesmas variáveis. Em Java é necessário identificar o tipo da variável sendo que em JavaScript, por ser interpretado, tipifica a variável no momento da alocação.

## INTRODUÇÃO A DADOS ESTRUTURADOS OU CONSTRUÍDOS

Nos algoritmos, assim como nas linguagens de programação, existe a possibilidade de se criar outros tipos de dados. Para melhor entender os termos técnicos utilizados pelos programadores, você deve entender as estruturas de dados das informações mais comuns utilizadas na computação. O tipo mais comum construído consiste na declaração de um conjunto de campos.

A estrutura construída **campo** é um conjunto de caracteres que contém uma informação. Exemplo: O valor do Dólar é de R\$ 3,19.

valorDólar	dataReferência
3,19	28/04/17

**A tabela 1:** Apresenta um exemplo de Campo de Dados

Onde valorDólar e dataReferência são campos, pois o valor do dólar irá modificar junto com o campo data, ou seja, a cada dia o dólar tem uma nova cotação.

O código abaixo exemplifica como você pode definir um campo em JavaScript. Observe que criamos o Campo com dois parâmetros, valorDolar e dataReferencia. O campo valorDolar está como uma constante, ou seja, não ocorrerá mudança nele. O campo dataReferencia é uma variável do tipo Date atribuindo automaticamente a data atual e local para a variável.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Tipo de Dados - Campo</title>
  </head>
  <body>
    <div id="area"></div>
    <script>
      var Campo = {
        valorDolar: 3.19,
        dataReferencia: new Date().toLocaleDateString()
      };
      document.getElementById("area").innerHTML =
        "O Dolar é R$ " + Campo.valorDolar.toFixed(2) +
        " em " + Campo.dataReferencia
      ;
    </script>
  </body>
</html>

```

O comando `document.getElementById` permite que o JavaScript modifique em tempo de execução a `<div>` "área" do HTML escrevendo a saída "O Dolar é R\$ 3.19 em ??/?/??" sendo as interrogações substituídas pelo data atual.

Observe como o mesmo código ficaria em Java. Java é uma linguagem orientada a objeto, assim, a estrutura campo será uma classe que será instanciada mais tarde. No NetBeans você vai criar um projeto Java e você deve gerar duas classes no projeto, uma Campo e outra TipoDados. A classe campo contém os atributos e estes serão instanciados em um objeto mais tarde.

```

package tipoDados;
import java.util.Date;
public class Campo {
  float valorDolar = 3.19f;
  Date data = new Date();
}

```

Em orientação a objeto, a classe Campo é apenas um modelo que servirá para criar inúmeros objetos. O código a seguir ajudará você a instanciar e testar a classe Campo. Observe que no exemplo a seguir criamos um objeto tipoDados do tipo Campo com o comando new.

```
package tipoDados;  
public class TipoDados {  
    public static void main(String[] args) {  
        Campo tipoDados = new Campo();  
        System.out.printf("O Dolar é R$ " +  
            td.valorDolar +  
            " em " + td.data  
        );  
    }  
}
```

Outra estrutura construída é o Registro, um conjunto de campos relacionados com um identificador comum.

Exemplo: O funcionário Claudiney Sanches, professor, matrícula número 5836, ganha R\$ 152,00 por hora e trabalhou 280 horas no mês.

Matricula	Nome	SalarioHora	HorasTrab
5836	Claudiney Sanches	52,00	280

A tabela 2 apresenta um exemplo de Registro de Dados  
Onde dizemos que os campos Matrícula, Nome, SalarioHora e HorasTrab se referem ao funcionário Claudiney Sanches. O código em HTML a seguir cria um registro de funcionário e a função Imprimir() escreverá na tag <p> (parágrafo) de forma dinâmica.

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="utf-8" />  
        <title>Tipos de Dados - Construídos</title>  
        <script>  
            var funcionario = {  
                Nome : "Claudiney Sanches",
```

```

        Matricula : 5836,
        SalarioHora: 152.00,
        HorasTrab: 280
    };
    function Imprimir() {
        document.getElementById("area").innerHTML =
        "<h1>Dados Funcionário</h1><br>" +
        "Nome: " + funcionario.Nome + "<br>" +
        "Matricula: " + funcionario.Matricula + "<br>" +
        "Valor Hora: R$" +funcionario.SalarioHora.toFixed(2)
        +"<br>" +
        "Horas Trabalhadas: " + funcionario.HorasTrab
        ;
    }
</script>
</head>
<body onload="Imprimir();">
<p id="area"></p>
</body>
</html>

```

Veja o mesmo código em Java acrescido de um método construtor que somente poderá ser executado uma vez no momento de criação do objeto Funcionário.

```

package testafuncionario;

public class Funcionario {
    private String Nome;
    private int Matricula;
    private float SalarioHora;
    private int HorasTrab;
    public Funcionario(String nome, int matricula,
                       float salarioH, int horasT){
        this.Nome = nome;
        this.Matricula = matricula;
        this.SalarioHora = salarioH;
    }
}

```

```
        this.HorasTrab = horasT;
    }
    public void Imprimir(){
        System.out.println("Dados Funcionário");
    }
}
```

---

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Variáveis em Javascript</title>
        <script>
            System.out.println("Valor Hora: R$ "+this.SalarioHora);
            System.out.println("Horas Trabalhadas: "+this.HorasTrab);
        }
    }
```

Na classe principal, você deve acrescentar o código que irá instanciar ou criar o objeto. Como a classe apresenta um método construtor, passamos a utilizá-lo para criar o objeto com os atributos definidos.

```
package testafuncionario;

public class TestaFuncionario {

    public static void main(String[] args) {
        Funcionario func = new Funcionario(
            "Claudiney Sanches",
            5836,
            152.00f,
            280
        );
        func.Imprimir();
    }
}
```

O objeto func é criado com os atributos Nome = "Claudiney Sanches", Matricula = 5836, SalarioHora = 152.00 e HorasTrab = 280. Quando a função Imprimir() é chamada ela irá colocar na janela de console todos os atributos do objeto func.

## TIPO ABSTRATO DE DADOS

É um conjunto de dados e operações nas quais apenas as operações definidas podem ser executadas sobre os dados. Podemos dizer que o Tipo Abstrato de Dados (TAD) oculta, como uma caixa preta, as informações sobre as estruturas de dados para o programador. Todos nós sabemos o que uma caixa preta faz em um avião, mas a maioria das pessoas não sabem **como** armazena ela faz. Veja a figura 1 que mostra a caixa preta do avião, e que nem tem a cor preta como muitos imaginam.



**Figura 1:** Caixa Preta/CVR records by ntsb.gov/aviation

O princípio de ocultar detalhes permite usar um tipo de dados sem se preocupar em conhecer como ele foi implementado internamente. Quando você for criar um TAD, será sua preocupação definir as estruturas de dados bem como as operações ou ações que ocorrem com a mesma. Você não deve permitir que se acesse as estruturas de dados de forma direta. O acesso deve ser apenas permitido por métodos ou funções que serão implementadas para tal fim. Idealmente, a implementação ou código de um TAD é invisível e inacessível aos usuários dela. Quando você for utilizar um TAD, bastará saber quais métodos estão disponíveis. A vantagem do uso de TAD é o reuso, facilidade de

manutenção e proteção adicional sobre manipulações inesperadas por parte de outros algoritmos. Quando usamos TAD, nossos programas ficam divididos em dois: programa usuários e implementação.

## ■ ESTRUTURAS DE DADOS BÁSICAS

O armazenamento com variáveis normalmente é suficiente para desenvolver programas que permitem a entrada de um único valor, porém existem casos em que você desejará armazenar um conjunto de valores. Por exemplo em um treino de Fórmula 1 você vai querer armazenar o tempo de cada volta do piloto de sua equipe, e ao final do treino, avaliar a média, a melhor volta e a pior. Para isso, vamos utilizar uma estrutura de dados indexada ou vetor. Neste tipo de dado, diversos valores são armazenados em uma estrutura mais complexa, cujos elementos são identificados com a ajuda de índices. O vetor ou **Array** é a estrutura mais simples pois apresenta apenas um índice que sempre iniciará em 0 e irá identificar o 1º elemento. A figura 2 apresenta um vetor e seu índice.



**Figura 2:** Índices de um Vetor

Para manipular um elemento no vetor, você vai precisar informar o nome do vetor e o índice do elemento. Na maioria das linguagens utilizamos os colchetes [ ] para indicar o índice do vetor. Veja um exemplo de uma estrutura de dados simples que armazena dados em um vetor.

```
<!DOCTYPE html>
<html>
  <head>
```

```

<meta charset="utf-8" />
<title>Vetor Simples - Meses</title>
<script>
    //Criar Vetor
    Meses = new Array();

    var dat = new Date();
    var m = dat.getMonth();

    //Armazenar a Lista
    Meses[0] = "Janeiro";
    Meses[1] = "Fevereiro";
    Meses[2] = "Março";
    Meses[3] = "Abril";
    Meses[4] = "Maio";
    Meses[5] = "Junho";
    Meses[6] = "Julho";
    Meses[7] = "Agosto";
    Meses[8] = "Setembro";
    Meses[9] = "Outubro";
    Meses[10] = "Novembro";
    Meses[11] = "Dezembro";

    //Imprimir Mes
    document.write("Estamos no mês de "+Meses[m]);
</script>
</head>
<body>
</body>
</html>

```

Veja o código novamente na linguagem Java. Observe que em Java, a declaração do vetor tem outra sintaxe. Para manipular a Data foi necessário acrescentar a biblioteca java.util.

```

package vetor;
import java.util.Date;

```

```

public class Vetor {

    public static void main(String[] args) {
        Date dt = new Date();
        int mes = dt.getMonth();
        String[] Meses = new String[12];
        Meses[0] = "Janeiro";
        Meses[1] = "Fevereiro";
        Meses[2] = "Março";
        Meses[3] = "Abril";
        Meses[4] = "Maio";
        Meses[5] = "Junho";
        Meses[6] = "Julho";
        Meses[7] = "Agosto";
        Meses[8] = "Setembro";
        Meses[9] = "Outubro";
        Meses[10] = "Novembro";
        Meses[11] = "Dezembro";
        System.out.println("Estamos no mês de "+Meses[mes]);
    }
}

```

Um atrativo dos vetores é que a indexação permite o acesso a qualquer elemento do vetor, independente da ordem e sem um custo extra de eficiência. Frequentemente tem operações que serão aplicadas a todos os elementos de um vetor. Neste caso, as estruturas de repetição podem percorrer todos os elementos. Veja o código que percorre o vetor efetuando a soma de todas as notas, atribuídas ao vetor.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Somar notas</title>
        <script>
            var soma = 0.0;

```

```

var notas = [3, 7, 2, 4.5, 8.2];
for (a = 0; a < notas.length; a++)
    soma += notas[a];
document.write(soma);
</script>
</head>
<body>
</body>
</html>

```

O código abaixo apresenta a implementação anterior em Java. Observe que existe outro método para definir um vetor já com valores definidos.

```

package somarnotas;
public class SomarNotas {
    public static void main(String[] args) {
        float soma = 0.0f;
        float notas[] = {3f,7f,2f,4.5f,8.2f};
        for(int a=0;a<notas.length;a++)
            soma += notas[a];
        System.out.println(soma);
    }
}

```

## PILHAS

No dia a dia, é comum empilhar objetos para serem retirados na ordem oposta da qual empilhamos. Um exemplo de uso é a pilha de pratos. Ao inserir um prato sobre o outro, tornamos acessível apenas o topo (**top**). A estrutura de dados Pilha implementa o conceito de o primeiro a entrar e o último a sair (**First-In, Last-Out** - FILO) e o último elemento a ser inserido na pilha é o primeiro a sair. Para demonstrar a estrutura de uma Pilha, você deve criar um vetor com uma função para inserir (**push**) dados no vetor, uma função para retirar o último dado que foi inserido (**pop**) e uma para ler. O código abaixo foi salvo com a extensão .js para designar um código abstrato de dados em JavaScript.

```

// JavaScript código de Dados Abstrato – Pilha.js
function Filo() {
    //Cria um Vetor Interno
    this.pilha = new Array();

    //Função para inserir no topo
    this.push = function (obj) {
        this.pilha[this.pilha.length] = obj;
    }

    //Função para retirar elementos do topo
    this.pop = function () {
        if (this.pilha.length > 0) {
            this.pilha.splice(this.pilha.length - 1, 1);
        } else {
            alert("Não consigo remover, pois, a pilha está vazia!");
        }
    }

    //Ler os elementos do topo
    this.ler = function () {
        if (this.pilha.length > 0) {
            var obj = this.pilha[this.pilha.length - 1];
            return obj;
        } else {
            return "Pilha vazia!";
        }
    }
}

```

Para testar o código abstrato que foi criado, você deve instancia-lo em um objeto. O exemplo a seguir cria o objeto livros por meio do comando new Filo();

```

<!DOCTYPE html>
<html>
    <head>

```

```

<meta charset="utf-8" />
<title>Criar uma Pilha</title>
<script src="Pilhas.js"></script>
<script>
    document.write("<h1>Livros para estudar</h1>")
    var livros = new Filo();
    livros.push("Sistemas de Banco de Dados: projeto, Implementação e gerenciamento");
    livros.push("SQL e Teoria Relacional");
    livros.push("Projeto e Modelagem de Banco de Dados");
    livros.push("Projeto de algoritmos: com implementação em Java e C++");
    document.write(livros.ler() + "<br />");
    livros.pop();
    document.write(livros.ler() + "<br />");
    livros.pop();
    document.write(livros.ler() + "<br />");
    livros.pop();
    document.write(livros.ler() + "<br />");
</script>
</head>
<body>

</body>
</html>

```

## FILAS

Todos nós já ficamos em uma fila, quer na escola, no cinema, para entrar no elevador ou sacar dinheiro no banco. O conceito de Filas em programação implementa o que você conhece tão bem. O primeiro a entrar na fila será o primeiro a sair ou ser atendido, e os demais elementos serão atendidos seguindo sequencialmente a ordem em que entraram na fila. Esse conceito é conhecido como **First-In, First-Out** ou simplesmente como FIFO. As filas ou **queues** são estruturas de dados importantíssimas para manter uma ordem cronológica dentro de um **software**. Nesta estrutura, temos dois métodos principais, um para inserir um elemento na fila e outro para remover o primeiro elemento da fila.



**Figura 3:** Funcionamento de um Fila

```
// JavaScript código Abstrato - Fila
function Fifo() {
    //Cria um Vetor Interno
    this.fila = new Array();

    //Função para inserir no fim da Fila
    this.inserir = function (obj) {
        this.fila[this.fila.length] = obj;
    }

    //Função para retirar primeiro da Fila
    this.remover = function () {
        if (this.fila.length > 0) {
            this.fila.splice(0, 1);
        } else {
            alert("Não consigo remover, pois, a fila está vazia!");
        }
    }

    this.ler = function () {
        if (this.fila.length > 0) {
            var obj = this.fila[0];
        }
    }
}
```

```

        return obj;
    } else {
        return "Fila vazia!";
    }
}
}

```

Observe que foram gerados dois arquivos, sendo o primeiro salvo como Fila.js e o segundo como Aula1FilaAtendimento.html. O primeiro trata de um código genérico ou abstrato que foi construído visando o reuso. O segundo utiliza o código abstrato para criar o objeto clientes que é do tipo Fifo que foi definido e criado no Fila.js.

```

<!DOCTYPE html>

<html>
    <head>
        <meta charset="utf-8" />
        <title>Fila de Atendimento</title>
        <script src="Fila.js"></script>
        <script>
            var clientes = new Fifo();
            clientes.inserir("1 - Claudiney Sanches");
            clientes.inserir("2 - Elisangela Cristina");
            clientes.inserir("3 - Rebecca");
            clientes.inserir("4 - Thais");
            clientes.inserir("5 - Maria");
            function sair() {
                document.getElementById("cxSaida").value = clientes.ler();
                clientes.remover();
            }
        </script>
    </head>
    <body>
        <form>
            <label>Número próximo cliente: </label>
            <input type="text" id="cxSaida" />
        </form>
    </body>

```

```
<button type="button" onclick="sair();">Chamar Próximo</button>
</form>
</body>
</html>
```

## LISTA LINEARES

Uma lista é uma coleção de elementos, do mesmo tipo de dados, dispostos em uma sequência determinada. Utilizamos listas em nosso dia a dia. Por exemplo, ao ir ao supermercado é comum encontrar pessoas com uma lista de compras que apresenta os itens desejados e a quantidade. Em uma escola, os professores mantêm uma lista de chamada com os nomes dos alunos. Em seu **smartphone**, encontramos a lista de contatos com os nomes de seus amigos.

O conjunto de dados armazenados em uma lista pode ser ordenado. No caso da lista de contatos, gostaríamos de ver os nomes em ordem alfabética para facilitar encontrar um nome. Na lista de compras do supermercado, é provável que mantenhamos a ordem de inserção ou cronológica.



leitura indicada

Para implementar uma Lista, Pilha e Fila em Java, recomendo ler o livro do Ziviani, Nívio; Projeto de algoritmos: com implementação em Java e C++; consultoria em Java e C++ de Fabiano Cupertino Botelho; São Paulo: Cengage Learning, 2015.

Para programar uma estrutura capaz de manipular tais informações, será necessário a criação de um vetor ou uma matriz. Essa técnica de implementação recebe o nome de Arranjo ou **Array**. Os vetores possuem limitações de espaço e muitas vezes será necessário manter variáveis para indicar qual o último elemento cadastrado. Outra solução seria implementar estruturas que consigam se interligar criando uma corrente ou elos, onde a primeira estrutura informa que está interligada à próxima, recebendo o nome de lista encadeada dinâmica. Listas são estruturas flexíveis que podem crescer ou diminuir de acordo com a

demandas, por isso se tornam adequadas para aplicações onde não é possível prever a utilização.

Existem várias estruturas de dados que podem implementar uma lista onde cada uma apresenta vantagens e desvantagens. Veja o exemplo de uma lista simples implementada com um vetor.

Amigos ← [Rebecca, Thais, Elisāngela, Claudiney, Pedro, Vinicius]

O exemplo em HTML5 e Javascript fica da seguinte maneira:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Vetor Simples</title>
    <script>
      //Criar a Lista
      Amigos = new Array();

      //Armazenar a Lista
      Amigos[0] = "Rebecca";
      Amigos[1] = "Thais";
      Amigos[2] = "Elisāngela";
      Amigos[3] = "Claudiney";
      Amigos[4] = "Pedro";
      Amigos[5] = "Vinicius";

      //Imprimir a Lista
      document.write("<h1>Lista de Amigos</h1>");
      NumElementos = Amigos.length;
      for (a = 0; a < NumElementos; a++) {
        document.write(Amigos[a] + "<br>");
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

É claro que a estrutura apresentada acima é simples, visando facilitar a compreensão. Uma implementação de lista tem que permitir inserir elementos em qualquer posição da lista, remover no início, no fim e em qualquer posição, além de permitir ordenar os resultados da lista. Assim, demonstraremos um código mais realista e bem mais complexo. Vale ressaltar que, para que você possa compreender o código apresentado, é interessante implementar apenas duas funções básicas a Inserir e Escrever.

```
// JavaScript código Abstrado de Lista.js
function Lista() {
    //Cria um Vetor Interno
    this.mlist = new Array();

    //Inserir o obj na última posição da lista
    this.inserir = function (obj) {
        this.mlist[this.mlist.length] = obj;
    }

    //Inserir o obj na posição indicada por pos na Lista
    this.inserirPosicao = function (obj, pos) {
        if (pos < this.mlist.length && pos >= 0)
            this.mlist.splice(pos, 0, obj);
        else
            alert("Posição Inválida");
    }

    //Retirar elemento do final da Lista
    this.remover = function () {
        this.mlist.splice((this.mlist.length - 1), 1);
    }

    //Retirar obj da Posição da Lista
    this.retirarPosicao = function (pos) {
        if (pos >= 0)
            this.mlist.splice(pos, 1);
        else
            alert("Posição Inválida");
    }
}
```

```

//Ler um obj na Posição
this.ler = function (pos) {
    return this.mlist[pos];
}

//Escrever todos os elementos
this.escrever = function () {
    texto = "<h2>Convidados cadastrados</h2>";
    var elementos = this.mlist.length;
    if (elementos != undefined) {
        for (count = 0; count < elementos; count++)
            texto += "<p>" + (count) + " - " + (this.mlist[count] + "</p>");
        document.getElementById("area").innerHTML = texto;
    }
}

//Ordenar a lista
this.ordenar = function () {
    this.mlist.sort();
}

//Ordenar em ordem decrescente
this.ordenarDecrescente = function () {
    this.mlist.sort();
    this.mlist.reverse();
}

```

O código abstrato acima foi construído por partes, sendo que para cada função desenvolvida, foi elaborado um código de teste. Assim, quando você for desenvolver um código abstrato ou biblioteca, faça uma função e teste. Quando ela estiver fazendo o que você quer, parta para a próxima. A seguir, veja o código que testa quatro funções apresentadas em Fila.js. Observe que esse é o primeiro html que apresenta css incorporado através da tag <style>.

```
<html>
<head>
    <meta charset="utf-8" />
    <title>Convidados Casamento</title>
    <style>
        body {
            background-color:rgba(255, 208, 174, 0.25);
        }
        h1 {
            border-radius: 60px;
            background-color:#ff6a00;
            height:110px;
            background-image:url("aula1_exemplo1.png");
            background-repeat: no-repeat;
            background-attachment: fixed;
            background-size: 5%;
            background-position-x:15px;
            background-position-y:45px;
            text-indent: 150px;
            color:#ffffff;
            font-size: 350%;
        }
        form {
            background-color:#ffffff;
            border-radius: 60px;
            border-style: solid;
            border-width: medium;
            border-color:#ff6a00;
        }
        table {
            margin-left:40px;
            margin-top:40px;
        }
        p{
            margin-left:40px;
            margin-top:20px;
        }
    </style>

```

```

.botao{
    background-color: #ff6a00;
    width: 250px;
    height:40px;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
}
.saidaDados{
    margin-top:20px;
    background-color:#ffffff;
    border-radius: 60px;
    border-style: solid;
    border-width: medium;
    border-color:#ff6a00;
    text-indent: 100px;
    color:rgb(48, 41, 35);
    font-size: 150%;
}
</style>
<script src="Lista.js"></script>
<script>
    var convidados = new Lista();
    function cadastrar() {
        var objConv = document.getElementById("nome");
        convidados.inserir(objConv.value);
        objConv.value = "";
        convidados.escrever();
        objConv.focus();
    }
    function cadastrarPos() {
        var posTexto = prompt("Qual posição?");
        var pos = parseInt(posTexto);

```

```

        var objConv = document.getElementById("nome");
        convidados.inserirPosicao(objConv.value, pos);
        objConv.value = "";
        convidados.escrever();
        objConv.focus();
    }
    function remover() {
        convidados.remover();
        convidados.escrever();
    }
    function removerPos() {
        var posTexto = prompt("Qual posição?");
        var pos = parseInt(posTexto);
        convidados.retirarPosicao(pos);
        convidados.escrever();
    }
}

</script>
</head>
<body>
<h1>Convidados para o casamento</h1>
<form>
    <table>
        <tr>
            <td><label>Nome: </label></td>
            <td><input type="text" id="nome" /></td>
        </tr>
    </table>
    <p>
        <input class="botao" type="button"
               onclick="cadastrar();"
               value="Cadastrar Convidado" />
        <input class="botao" type="button"
               onclick="remover();"
               value="Remover Convidado" />
    </p>
    <p>

```

```

<input class="botao" type="button"
      onclick="cadastrarPos();"
      value="Cadastrar Posição" />
<input class="botao" type="button"
      onclick="removerPos();"
      value="Remover Posição" />
</p>

```

---

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Variáveis em Javascript</title>
    <script>
      </html>

```

## ■ ESTRUTURAS DE DADOS AVANÇADO

Apresentaremos a estrutura de dados que armazena Tabela como sendo o conjunto de registros sobre um determinado assunto.

- Exemplo: Tabela de funcionários

<b>Matricula</b>	<b>Nome</b>	<b>SalarioHora</b>	<b>Horas Trab</b>
5836	Claudiney Sanches	52,00	280
5837	Elisângela Cristina	29,00	240
5838	Tháis	18,00	240
5839	Rebecca	20,00	180

**Tabela 3:** Apresenta um exemplo de Tabela com seus vários registros.

É possível você implementar tal estrutura manualmente com a utilização de matrizes. Uma matriz é um vetor multidimensional com posição de linha para o registro e a coluna para o campo. A implementação de uma matriz é possível na maioria das linguagens de programação. Assim, vamos exemplificar em HTML uma matriz e em Java um vetor de objetos.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Matrizes</title>
    <script>
      var meusFuncionarios = [
        [5836,"Claudiney Sanches", 59.00, 280],
        [5837,"Elisangela Cristina", 29.00, 240],
        [5838,"Thais", 18.00, 240],
        [5839,"Rebecca", 20.00, 180]
      ];
      count = 0;
      function lerFuncionarios() {
        if (count < meusFuncionarios.length) {
          document.getElementById("cMatricula").value =
            meusFuncionarios[count][0];
          document.getElementById("cNome").value =
            meusFuncionarios[count][1];
          document.getElementById("cVHora").value =
            meusFuncionarios[count][2].toFixed(2);
          document.getElementById("cHTrab").value =
            meusFuncionarios[count][3];
          count++;
        } else {
          alert("Não existe mais registros para exibir!");
        }
      }
    </script>
  </head>
  <body>
    <p>
      <label>Matricula:</label>
      <input type="text" id="cMatricula" />
      <br />
      <label>Nome:</label>
      <input type="text" id="cNome" />

```

```

<br />
<label>Valor Hora R$ </label>
<input type="text" id="cVHora" />
<br />
<label>Quantidade de Horas: </label>
<input type="text" id="cHTrab" />
<br />
<button onclick="lerFuncionarios();">Ler Próximo Registro</button>
</p>
</body>
</html>

```

A estrutura apresentada pôde ser implementada em uma matriz em JavaScript e o HTML5 apresentou os dados através de **input** do tipo **text**. Ao ser pressionado o botão Ler Próximo Registro, o html, por meio do evento **onclick**, chama a função do JavaScript lerFuncionarios() e assim, irá ler os dados previamente armazenados na matriz e exibi-los nos campos **input** criados no html. Dessa forma, as duas linguagens interagem, sendo o html a saída de dados e o JavaScript responsável por modificar o html de forma dinâmica.

Vale observar que em Java, você pode criar uma estrutura com matriz, mas visando a manutenção e o reuso do código, podemos implementar um vetor de objetos. A seguir, apresentaremos um exemplo de código com vetores de objetos. Observe como esse tipo de estrutura eleva o grau de complexidade, dificultando a leitura e compreensão. A seguir, veja o código da classe Funcionarios. Para implementar esse projeto, no **NetBeans** crie um novo projeto Java Aplicação de nome Matriz e não selecione a opção "Criar Classe Principal". Adicione a classe Funcionario como o código abaixo.

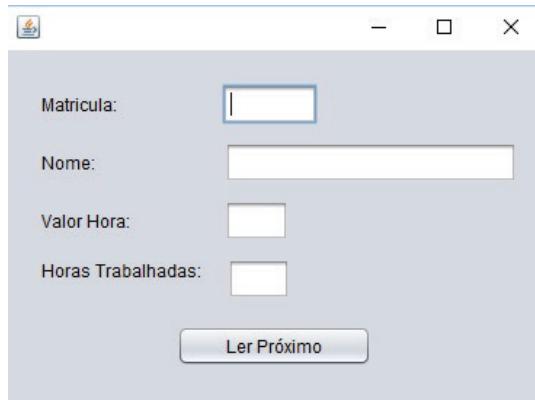
```

package matriz;

public class Funcionarios {
    public String Nome;
    public int Matricula;
    public float ValorHora;
    public int HorasTrab;
}

```

Em particular, esse exemplo utiliza um JFrame, ou seja, uma janela com interface gráfica. A figura 4 apresenta o JFrame utilizado. Para criar um JFrame no NetBeans vá na pasta Pacotes de Códigos-fonte e sobre o pacote matriz clique com o botão auxiliar do mouse em Novo Form JFrame e dê o nome de Visualizar. Desenhe a interface com **label** e caixas texto e um botão como apresentado na figura 4.



**Figura 4:** Formulário em Java SE utilizando JFrame

Ao dar um duplo clique sobre o botão, o **NetBeans** irá mudar do modo Projeto para o modo Código Fonte. Neste modo, você poderá editar o código fonte, acrescentando as linhas abaixo. Note que algumas linhas não poderão ser editadas no modo Código Fonte, sendo permitida a edição no modo Projeto. Para não lhe confundir, parte do código não editável foi suprimido e está na cor acinzentada.

```
package matriz;

public class Visualizar extends javax.swing.JFrame {
    int count = 0;
    Funcionarios[] meusFuncionarios = new Funcionarios[4];

    public Visualizar() {
        initComponents();
        Funcionarios f1 = new Funcionarios();
        f1.Nome = "Claudiney Sanches";
        f1.Matricula = 5386;
        f1.ValorHora = 59.0f;
        f1.HorasTrab = 280;
```

```

Funcionarios f2 = new Funcionarios();
f2.Nome = "Elisangela Cristina";
f2.Matricula = 5387;
f2.ValorHora = 29.0f;
f2.HorasTrab = 240;
Funcionarios f3 = new Funcionarios();
f3.Nome = "Thais";
f3.Matricula = 5388;
f3.ValorHora = 18.0f;
f3.HorasTrab = 240;
Funcionarios f4 = new Funcionarios();
f4.Nome = "Rebecca";
f4.Matricula = 5389;
f4.ValorHora = 20.0f;
f4.HorasTrab = 280;
meusFuncionarios[0]=f1;
meusFuncionarios[1]=f2;
meusFuncionarios[2]=f3;
meusFuncionarios[3]=f4;
}

```

+ Generated Code {92 Linhas Suprimidas}

```

private void btnProximoActionPerformed(
    java.awt.event.ActionEvent evt) {
    txtNome.setText(meusFuncionarios[count].Nome);
    String temp =
        Integer.toString(meusFuncionarios[count].Matricula);
    txtMatricula.setText(temp);
    temp = Float.toString(meusFuncionarios[count].ValorHora);
    txtValor.setText(temp);
    temp = Integer.toString(meusFuncionarios[count].HorasTrab);
    txtHoras.setText(temp);
    if (count<3)
        count++;
    else
        count=0;
}

```

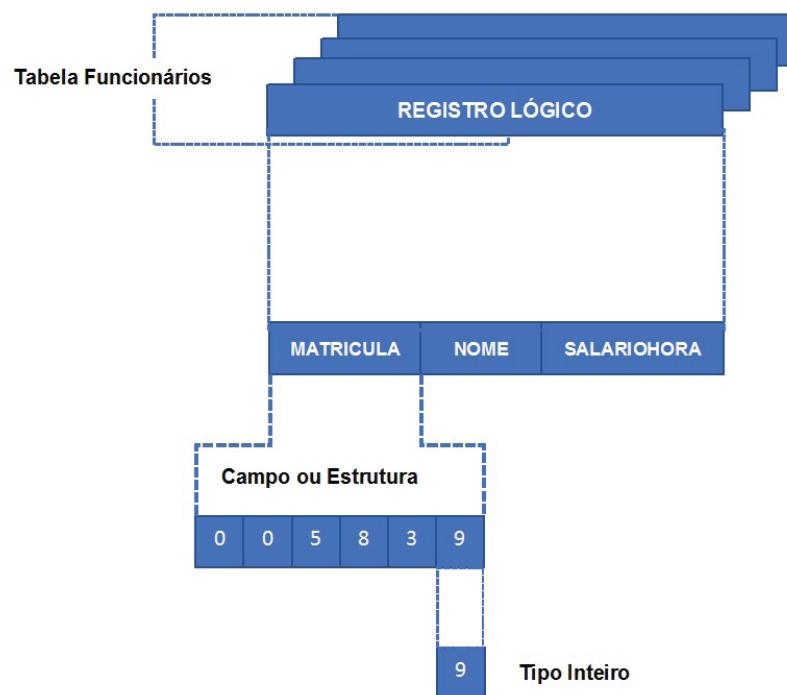
```

    }
    //só modifique pelo modo Projeto
    public static void main {31 linhas suprimidas}

    // Variables declaration - do not modify
    private javax.swing.JButton btnProximo;
    private javax.swing.JLabel lHoras;
    private javax.swing.JLabel lMatricula;
    private javax.swing.JLabel lNome;
    private javax.swing.JLabel lValor;
    private javax.swing.JTextField txtHoras;
    private javax.swing.JTextField txtMatricula;
    private javax.swing.JTextField txtNome;
    private javax.swing.JTextField txtValor;
    // End of variables declaration
}

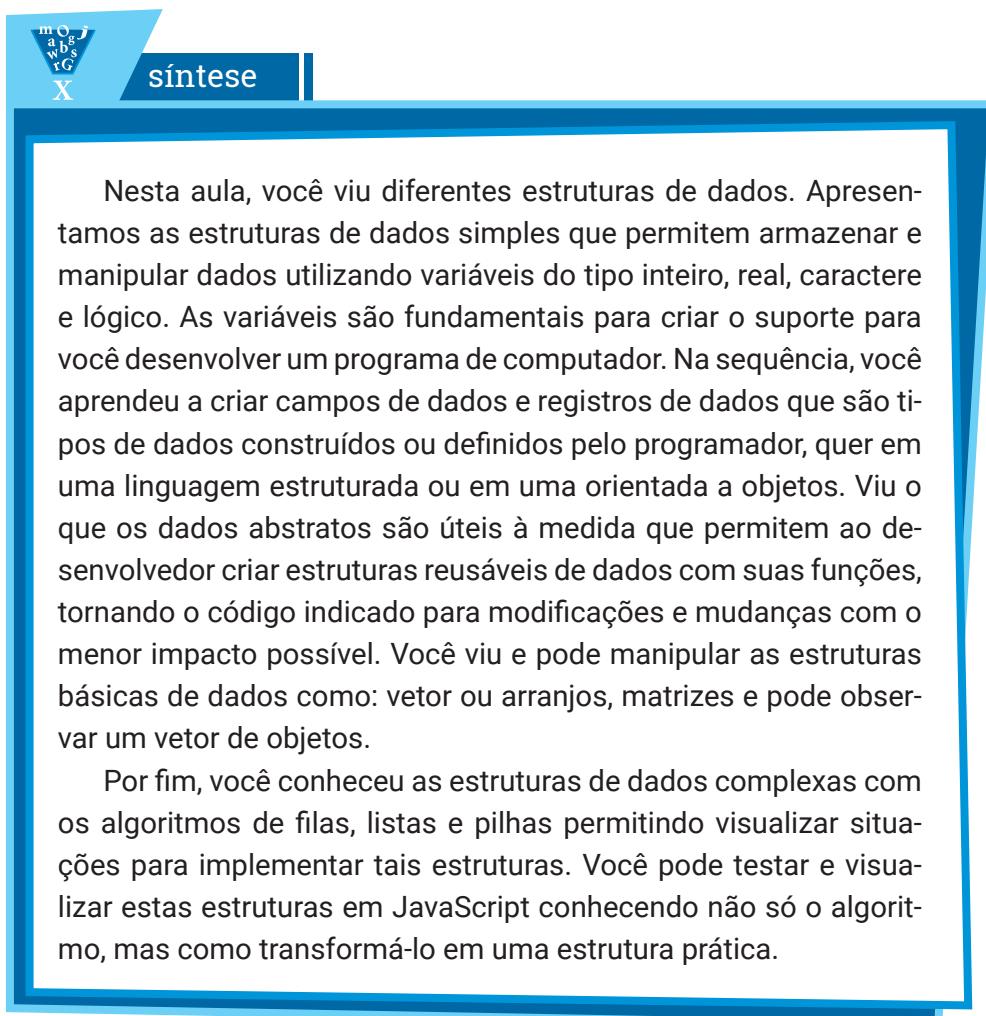
```

Outra estrutura lógica da informação que devemos aprender é a Tabela. A tabela é encapsulada, ou seja, contém registro lógico sendo que os registros lógicos contém campos e os campos contém dados variáveis, como representado na figura 5.



A figura 5 apresenta a estrutura de um Registro Lógico ou Tabela.

A estrutura de dados tabela será estudada e apresentada nas próximas aulas em união aos Sistemas de Gerenciamento de **Banco de Dados**.



The diagram illustrates a logical record or table structure. At the top left is a small icon containing letters like 'm', 'O', 'g', 'J', 'a', 'b', 's', 'w', 'r', 'G'. To its right is a blue bar with the word 'síntese' in white. Below this is a large blue-bordered box containing text. To the right of the box is a light blue callout box with the title 'Banco de Dados' and a definition.

Nesta aula, você viu diferentes estruturas de dados. Apresentamos as estruturas de dados simples que permitem armazenar e manipular dados utilizando variáveis do tipo inteiro, real, caractere e lógico. As variáveis são fundamentais para criar o suporte para você desenvolver um programa de computador. Na sequência, você aprendeu a criar campos de dados e registros de dados que são tipos de dados construídos ou definidos pelo programador, quer em uma linguagem estruturada ou em uma orientada a objetos. Viu o que os dados abstratos são úteis à medida que permitem ao desenvolvedor criar estruturas reusáveis de dados com suas funções, tornando o código indicado para modificações e mudanças com o menor impacto possível. Você viu e pode manipular as estruturas básicas de dados como: vetor ou arranjos, matrizes e pode observar um vetor de objetos.

Por fim, você conheceu as estruturas de dados complexas com os algoritmos de filas, listas e pilhas permitindo visualizar situações para implementar tais estruturas. Você pode testar e visualizar estas estruturas em JavaScript conhecendo não só o algoritmo, mas como transformá-lo em uma estrutura prática.

### Banco de Dados

são um conjunto de arquivos relacionados entre si com registros sobre pessoas, lugares ou coisas.

## Atividades

- Crie um arquivo HTML que, utilizando JavaScript, armazene a base e a altura de um retângulo e calcule sua área e perímetro apresentando os resultados no corpo do HTML

---

---

---

---

---

---

---

---

---

---

- 02.** Crie um HTML que permita criar uma lista de convidados para um churrasco. A lista de convidados deve sempre ficar visível no corpo da página. A página deve ter os botões de Inserir, Ordenar, Remover, Calcular Total de Carne. A pessoa deve inserir um nome em uma caixa de texto e clicar em Inserir o nome que será armazenado em um vetor. Ao clicar em ordenar, o vetor deve ser colocado em ordem alfabética. O botão Remover deve perguntar qual a posição a ser removido e eliminar o nome da pessoa da lista de convidados. O botão Calcular o total de carne deve contar o total de convidados e calcular para cada convidado 350g de carne bovina, 120g de linguiça e 30g de frango. Ao final do cálculo deve converter o resultado para quilos e exibir para o usuário no corpo da página.

---

---

---

---

---

---

---

---

---

---

---

---

# AULA 2

## INTRODUÇÃO A SISTEMAS DE BANCO DE DADOS

### NESTA AULA

- » A importância de se estudar e aprender Banco de Dados.
- » Raízes históricas.
- » Problemas de gerenciamento de dados do sistema de arquivos.
- » Modelos de dados, sua importância e diferenças.
- » Regras de negócios e evolução dos modelos de dados.

### I METAS DE COMPREENSÃO

- » Reconhecer a importância de estudar e aprender Banco de Dados;
- » Conhecer as raízes históricas e as diferenças entre sistemas de arquivos e SGBD, quais problemas os sistemas de arquivos tinham e como os SGBD abordam tais problemas;
- » Identificar os tipos e abstrações de dados;
- » Conhecer os modelos de dados, suas diferenças e importância;
- » Compreender os diagramas que apresentam as regras de negócios;
- » Conhecer as arquiteturas de SGBD e os Modelos Conceitual, Lógico e Físico.

### I APRESENTAÇÃO

Nosso objetivo nesta aula é tratar da importância do banco de dados e apresentar os principais termos da área. Vamos entender as raízes históricas dos bancos de dados e os problemas enfrentados com os sistemas de arquivos. O estudo dos principais conceitos de banco de dados o ajudará a se tornar um melhor administrador de banco de dados (DBA). Também iniciaremos a modelagem de negócios da empresa, criando diagramas que vão nos ajudar a compreender como a empresa funciona e suas necessidades de dados.

# A IMPORTÂNCIA DE ESTUDAR E APRENDER BANCO DE DADOS

## Banco de Dados

conjunto de dados relacionados capazes de apresentar uma informação

## Metadados

dados sobre os dados, conjunto de descrição dos dados e relacionamentos

## Inconsistência de dados

as bases não se mantêm síncronas

Os bancos de dados estão presentes nas vidas das pessoas. É comum, ao ir em uma loja para adquirir um produto, passar por um processo para ser inserido um cadastro de cliente. Quando você pergunta ao vendedor o preço e disponibilidade de um produto, este ao consultar um terminal para lhe responder, está consultando as informações armazenadas em um **banco de dados**. Assim, podemos definir bancos de dados como um conjunto de dados relacionados capazes de apresentar uma informação. Os dados são fatos brutos de interesse do usuário. Os bancos de dados são as estruturas de dados computacionais que permitem o armazenamento de dados e metadados. Os bancos de dados atendem à uma comunidade distinta de usuários, e são mecanismos de armazenamento de registros em tabelas. Já os **metadados** são dados sobre os dados, ou seja, fornecem uma descrição das características dos dados e do conjunto de relacionamentos que ligam esses dados ao banco de dados. Por exemplo, em um campo podemos ter como metadados os nomes do campo e o tipo de dados que ele armazena além de informações sobre se ele aceita valores nulos.

Houve um tempo em que os dados eram armazenados em diferentes bancos de dados, como arquivos, planilhas e pequenos arquivos de textos espalhados em disquetes e fitas magnéticas. Apesar de ser um avanço para aquele tempo, ocorria muita perda de dados ocasionando transtornos para as empresas. Muitos problemas estavam relacionados à redundância e **inconsistência de dados**. Outro grande desafio era a dificuldade de se manter diferentes bases de dados atualizadas e seguras, além de problemas de arquivos serem corrompidos e estarem sujeitos à degradação física. Outra dificuldade era que as informações entre os departamentos estavam isoladas e não se relacionavam, sendo difícil integrá-las e utilizá-las para auxiliar na tomada de decisão.

Vamos ilustrar alguns dos problemas mencionados com um caso fictício. Imagine que o departamento de vendas apresentasse uma planilha

eletrônica com o nome dos produtos comercializados pela empresa, bem como o nome dos vendedores, a lista de clientes e os pedidos feitos pelos clientes. O departamento financeiro da empresa tivesse a lista de clientes, o valor faturado, e a situação do faturamento. O primeiro problema era manter os dados dos clientes atualizados em ambos os departamentos. É provável que, com o decorrer do tempo, as atualizações que ocorressem em um departamento em uma das planilhas, não fossem visualizadas por outro departamento. Este problema de se ter dados em duas bases se chama **redundância de dados** e a falta de sincronismo entre as bases se chama de inconsistência de dados, pois os dados alterados ou cadastrados em uma base não sincronizam com a outra base. Quanto à integração, podemos ilustrar pela busca de dados entre ambos os bancos de dados, como por exemplo, qual foi o valor efetivamente recebido pelas vendas de um vendedor da empresa, ou ainda a busca de quais vendedores apresentam o maior índice de clientes inadimplentes. Essas informações podem mudar as comissões do departamento de vendas que, ao trabalhar de forma isolada, poderia apresentar um vendedor como “o melhor vendedor da empresa” por ser o que mais faturou. Ao se cruzar informações, nota-se não apenas o faturamento bruto, mas o recebido ponderando entre vendas efetuadas e clientes que pagam a empresa.

### Redundância de dados

ter dados em diferentes locais



pense nisso

Qual o Banco de Dados mais utilizado no mercado? Qual o SQBD que devo estudar? Quanto ganha um DBA? Tem diferença de salário se eu escolher um determinado SGBD?

Redundância de dados é quando um determinado campo está armazenado diversas vezes. Podemos ter uma redundância controlada à medida que o sistema conhece que os dados estão armazenados de forma duplicada, e ele é responsável por sincronizar esses dados. Normalmente, quando integramos dois sistemas, ocorre redundância controlada. O outro tipo de redundância é a não controlada, na qual o usuário é o responsável por sincronizar os dados. Esta abordagem apresenta problemas, uma vez que o usuário falha em sincronizar os dados criando inconsistência. Para solucionar isso, surgiu o compartilhamento de dados.

## SGBD

Sistema de gerenciamento de banco de dados, conjunto de aplicações de software que permitem gerenciar os bancos de dados em 3 aspectos de visões: externa, conceitual e interna.

## Integração de dados

busca entre dados de diferentes departamentos ou base de dados

Para evitar concorrência de recursos e acesso simultâneo aos dados, é necessário um software para gerenciar este acesso, sendo esta uma das tarefas atribuídas aos Sistemas de Gerenciamento de Banco de Dados (**SGBD**). O SGBD é um conjunto de software criado para facilitar, criar, manipular, controlar acesso e segurança de um Banco de Dados.

## VANTAGENS DO USO DE UM SGBD

O SGBD serve como um mediador entre o usuário e o banco de dados. A presença de um SGBD deve oferecer algumas vantagens importantes. Ele deve:

- proporcionar abstração isolando o usuário dos detalhes internos das estruturas de dados;
- aprimorar o compartilhamento, tornando fácil e rápido o acesso aos dados;
- proporcionar independência dos dados em relação a estratégia de acesso;
- facilitar que diversas aplicações accessem a base de dados;
- bloquear o acesso à sua estrutura interna;
- permitir o acesso somente por meio de uma interface segura aplicando uma política de privacidade e segurança de dados;
- melhorar a **integração de dados** por diversos departamentos da empresa;
- minimizar a inconsistência de dados;
- aumentar a produtiva do usuário final e
- aprimorar a tomada de decisão.

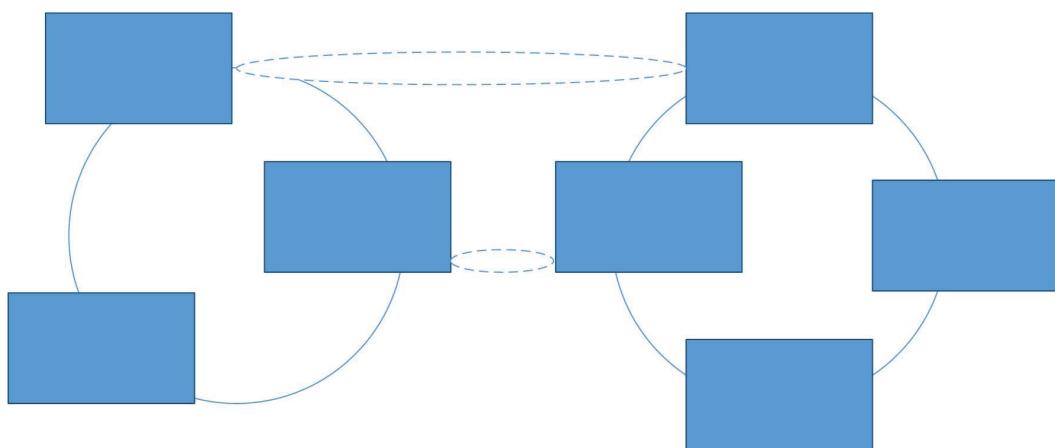
Essas são apenas algumas das vantagens do uso de um SGBD. Quanto mais você estuda e conhece o SGBD, mais vantagens lhe são apresentadas.

## ■ RAÍZES HISTÓRICAS

Na década de 1960, diferentes empresas, de olho nos custos dos serviços de anexar e indexar arquivos, iniciaram pesquisas para automação de escritórios. Destas pesquisas, saíram dois sistemas de

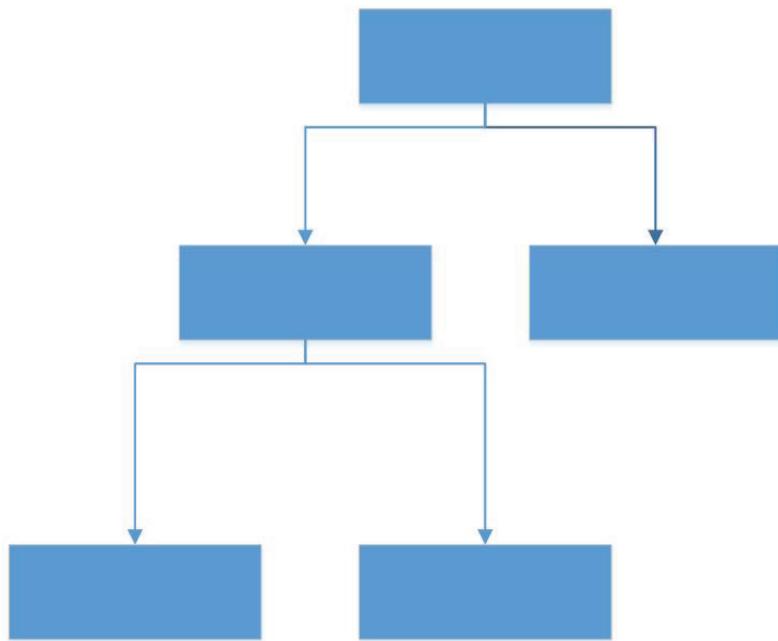
manipulação de dados: CODASYL e IMS. Em ambos os sistemas, o acesso era feito por meio de ponteiros exigindo reescrita e programação em baixo nível.

Em 1964, Charles Bachman, da General Electric, propôs um modelo de rede com registros de dados interligados, formando conjuntos de dados que se cruzavam, conforme ilustrado na figura 1. Esta pesquisa formou a base do Grupo de Tarefas da Base de Dados ou CODASYL. Neste modelo há mais de um caminho em que um determinado dado pode ser acessado. As relações se dão na proporção de um-para-muitos entre membro-proprietário. Um membro pode ter mais de um proprietário. Para exemplificar, imagine que existam 7 projetos representados na figura 1 como sendo um retângulo. Esses projetos foram divididos em dois departamentos, representados pelos círculos. Em determinado instante, um departamento da empresa tem que colaborar com outro, criando assim uma área de união representada pela elipse pontilhada.



**Figura 1:** Modelagem de Dados CODASYL

Em 1965, a Divisão Espacial da Aviação da América do Norte e a IBM desenvolveram uma segunda abordagem baseada em um modelo hierárquico onde os dados são representados como uma estrutura de árvore hierarquia de registros, como mostrado na ilustração da Figura 2. Em 1969 a IBM apresentou um produto baseado neste modelo, o Sistema de Gestão da Informação (IMS). Neste modelo os dados são organizados de cima para baixo e mantém uma relação lógica onde um determinado nó não pode existir sem o nível anterior, mantendo uma relação de dependencia.



**Figura 2:** Modelagem de Dados IMS

Em 1970, o matemático Edgar Frank Ted Codd, funcionário da IBM, escreveu um artigo com um tipo de notação matemática que não foi levada a sério. Então, Ted organizou um simpósio e conseguiu resumir cinco páginas de programação complicada em uma única linha chamando a atenção da IBM. Codd imaginava um sistema de tabelas em que o usuário poderia acessar os dados com comandos em inglês. A partir deste momento, a IBM criou o projeto Sistema R que levaria a criação da **SQL** (*Structured Query Language*, ou Linguagem de Consulta Estruturada que mais tarde se tornou padrão na indústria para banco de dados relacionais), que na época se chamava de “SEQUEL”. Os objetivos do projeto Sistema R era criar um sistema de banco de dados relacionais e desenvolver mais tarde um produto. Sem perceber o valor de mercado da pesquisa do grupo, a IBM permitiu que a equipe publicasse seus trabalhos. Outro pesquisador que contribuiu para a criação e aperfeiçoamento dos atuais bancos de dados foi Peter Chen que publicou um artigo em 1976 sobre o modelo entidade-relacionamento (MER) com o objetivo de criar uma notação para que o desenvolvedor focasse na utilização dos dados, sem se preocupar com as estruturas lógicas.

Entre os leitores dessas pesquisas estava o microempresário Larry Ellison que, em 1977, fundou a **SDL** (*Software Development Laborato-*

## SQL

Linguagem de Consulta Estruturada

ries) que acabou recrutando os programadores do Sistema R e da Universidade da Califórnia. Em 1979, Ellison colocou no mercado o primeiro banco de dados relacional com base no SQL. No inicio dos anos 80, quase todas as implementações de bancos de dados usavam a abordagem de rede ou hierárquica, sendo o de maior sucesso o IMS que, mesmo depois de 20 anos, ainda pode ser encontrado em muitas organizações. Em 1982, a empresa mudou o nome para Oracle e lançou sua primeira versão portátil do banco de dados, atingindo um faturamento de 5 milhões de dólares naquele ano. Impelida pela concorrência, a IBM finalmente lança o SQL/DS que posteriormente passou a chamar DB2.

Os bancos de dados evoluíram e hoje são chamados de SGBD (Sistema de Gerenciamento de Bancos de Dados). O SGBD é um conjunto de aplicações de software que permite gerenciar os bancos de dados nos aspectos das 3 visões: externa, conceitual e interna.

Com o aumento do uso dos computadores pessoais e o *feedback* dos usuários, os SGBDs foram refinados. No final da década de 1980, os sistemas relacionais passam há apresentar maturidade tecnológica e a ter um desempenho aceitável. Neste mesmo período, o banco de dados orientado a objeto passa a ter suas primeiras distribuições e surgem combinações entre banco de dados orientados a objetos e relacional. Além disso, a linguagem para manipulação de dados SQL passa a ser padronizada como ANSI (*American National Standards Institute*).

No início de 1990, o modelo cliente-servidor (client-server) passa a ser o mais utilizado no desenvolvimento de software. Na metade da década, os SGBDs têm que lidar com Bibliotecas digitais, hipermídia, multimídia, animações e vídeos sob demanda. Além disso, os dados armazenados precisavam ser explorados e novas formas de buscas são exploradas utilizando algoritmos de aprendizagem e redes neurais - *Data Mining*. Outros bancos de dados são criados como GIS (Sistema de Informações Geográficas), Meteorológicas, Física de Alta Energia (HEP). No final da década surgem o **Web Information Integration System** (WIIS) que trata dos dados provenientes de websites, **Data Warehouse** que extrai dados de grandes fontes de armazenamento.

Em maio de 2017, o SGBD mais utilizado no mundo foi o Oracle seguido pelo MySQL, SQL Server da Microsoft, PostgreSQL (*open souce*), MongoDB (não relacional) e DB2 da IBM.

**Web Information Integration System**

trata dos dados provenientes de websites

**Data Warehouse**

extrai dados de grandes fontes de armazenamento

## DIFERENÇAS ENTRE SGBD E SISTEMAS DE ARQUIVOS

Há alguns anos atrás, os gerentes de empresas eram capazes de rastrear qualquer transação comercial por meio de arquivos físicos. Estes eram organizados com base em armários de metal, com gavetas e pastas suspensas etiquetadas. À princípio, o conteúdo de cada pasta era logicamente organizado. Assim, o departamento de recursos humanos determinava que cada gaveta do armário seria utilizada para um departamento e cada funcionário do departamento ganhava uma pasta que continha todos os documentos desse funcionário. Enquanto o sistema se mantivesse pequeno e com poucos relatórios, o sistema manual executava seu papel. Contudo, com o crescimento da empresa e o aumento da demanda por relatórios, ficava mais difícil rastrear dados, tornando essa tarefa enfadonha e não confiável.

O início da computação apresenta uma forma eletrônica de armazenar e lidar com um volume de dados que crescia a cada dia. Contudo, os problemas mais comuns encontrados à princípio foram: redundância, compartilhamento, controle de acesso, esquematização, isolamento e abstração, suporte a visões e falta de autodescrição. Os arquivos eletrônicos apresentavam todos esses problemas. Assim o SGBD surge como uma ferramenta que cria uma camada entre o sistema de arquivos e o usuário solucionando os principais problemas do sistema de arquivos. O SGBD deve garantir que a informação tenha o mínimo de redundância de dados, deve estar disponível para qualquer número de usuários, deve ter o acesso por meio da identificação de quem acessou e quais ações realizou, o armazenamento deve seguir uma padronização visando facilitar o entendimento e terminologia entre diversas aplicações. O SGBD deve separar de forma distinta as estruturas de dados, os dados e o acesso dos programas aos dados. Deve garantir o isolamento e abstração dos dados além de manter uma estrutura auto descritiva dos dados. Por fim, deve dar a cada usuário uma visão diferente do banco, permitindo ver e manipular apenas alguns campos.

O SGBD surgiu para atender a necessidade de controle sobre um grande volume de dados. Deve proporcionar um ambiente seguro e adequado ao armazenamento e recuperação de dados. Seu objetivo é, de forma compacta, retirar da aplicação a responsabilidade de geren-

ciar o armazenamento de dados. Assim, a interface deve criar maneiras para a inserção, manipulação e o armazenamento de dados.

Existem atualmente diferentes arquiteturas de SGBD. A mais conhecida é ANSI/SPARC que se fundamenta em três níveis de abstrações.

Os esquemas são divididos em:

- ➊ nível externo – descreve o modo pelo qual os dados são vistos pelos usuários. Cada visão descreve quais porções dos dados um usuário ou grupo terá acesso;
- ➋ nível conceitual – descreve a estrutura de dados e seus relacionamentos, não descreve como os dados são fisicamente armazenados;
- ➌ nível interno ou lógico – descreve a estrutura interna do sistema, como ele irá armazenar os dados. É a descrição mais próxima de como os dados serão armazenados.

## ■ PROBLEMAS DE GERENCIAMENTO DE DADOS DO SISTEMA DE ARQUIVOS

O sistema de arquivo foi a primeira automação em relação ao sistema manual. Ele serviu como sistema de gerenciamento de arquivos por mais de duas décadas. Muitos problemas e limitações surgiram e tornaram evidentes a necessidade de uma nova abordagem. Um dos problemas era a necessidade de programação para buscas simples, sendo necessário especificar o que deveriam buscar e como deveriam fazer. Atualmente os SGBD utilizam uma linguagem de manipulação de dados que especifica apenas o que ele deve fazer sem especificar como fazer. No sistema de arquivos, é necessário tempo para programar novas consultas, sendo que normalmente o usuário não tem esse tempo. Também alterar as estruturas de dados em um sistema que está operacional é difícil em um sistema de arquivos. Por exemplo, imagine que se perceba a necessidade de acrescentar um campo em um arquivo cliente. Será necessário fazer um programa que leia o arquivo

cliente, copie todos os campos um-por-um para o arquivo cliente2 e acrescente o novo campo. Ao finalizar, o arquivo cliente será apagado e deve-se repetir o processo para que os dados do arquivo cliente2 sejam copiados para o arquivo cliente. Além disso, deve-se garantir que todos os programas que utilizam o arquivo cliente estejam compatíveis com a mudança, causando momentos de instabilidade nos programas que utilizam o sistema de arquivo. Outro problema é que com o aumento do número de arquivo, aumenta a dificuldade de administrar os diversos programas gerados para manipular tais arquivos. Muitos desses sistemas de arquivos acabam sendo gerados devido à impossibilidade de gerar consultas *ad hoc*, exigindo que se criem programas para relatórios sempre crescentes. Por último, podemos apresentar, como fator crítico, a dificuldade de implementar segurança no sistema de arquivo.

## ■ MODELOS DE DADOS, SUA IMPORTÂNCIA E DIFERENÇAS

### Modelagem de dados

criar estruturas computacionais capazes de representar computacionalmente o mundo real

**Modelagem de dados** é observar parte do mundo e criar estruturas de dados que são capazes de representar computacionalmente o mundo real. É o estudo das informações representando-as em modelos em um contexto. A modelagem de um banco de dados deve omitir dos usuários como as estruturas de dados serão armazenadas e processadas, tratando os dados em uma camada de abstração. A modelagem de um banco de dados pode ser utilizada para descrever estruturas lógicas e físicas. Ao comparar o modelo de um banco de dados com um modelo orientado a objeto, nota-se que os diagramas de classes apresentam não somente as informações dos dados como contemplam as ações que estes sofrem, como métodos de acesso, recuperação, processamento e outros. A modelagem de dados separa os dados dos métodos o que facilita a compreensão.

### Análise Orientada a objetos

contempla modelagem de dados e modelagem orientada a objetos

A **Análise Orientada a Objetos** (AOO) contempla ambos os modelos, misturando e integrando a modelagem de dados com a modelagem orientada a objetos. Este processo chama-se *two fase commit* que uti-

liza padrões de modelos de dados (**patterns**) e o modelo de dados Entidade-Relacionamento (ER). Para isso, o analista tem que entender e dominar técnicas de modelagem de dados e aprender a lidar com os modelos orientados a objetos que apresentam novamente os mesmos dados. O analista deve procurar encontrar modelos que representam da melhor forma possível os dados de uma realidade.

O objetivo ao criar um modelo de dados é garantir que todos os dados que uma aplicação precisa e manipula estejam representados com precisão. Deve facilitar a revisão pelos usuários finais e apresentar detalhes suficientes para que o Administrador de Banco de Dados (**DBA**) consiga construí-lo.

Caso o processo de modelagem seja mal feito, é provável que se gaste mais tempo para criar o banco de dados e que aumente o retrabalho no banco de dados. Ao elaborar uma modelagem rápida, o DBA acabará omitindo dados necessários ou criando inconsistência e isso levará, com o tempo, à impossibilidade de acomodar mudanças e manutenção excessiva.



dica

O modelo de dados deve ser simples para se comunicar com o usuário final mas deve ser detalhado para que o DBA consiga criar as estruturas físicas em um SGBD. O modelo mais comum para a construção de banco de dados é o Entidade-Relacionamento.

**Patterns**

padrões de modelos de dados

**DBA**

Administrador de Banco de Dados

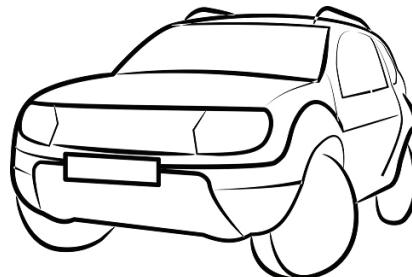
## ■ REGRAS DE NEGÓCIOS E EVOLUÇÃO DOS MODELOS DE DADOS

### ABSTRAÇÃO DE DADOS

Um SGBD é composto por estruturas de dados que são uma coleção de arquivos relacionais e por diferentes programas de manipulação de dados. Esses programas permitem que o usuário acesse arqui-

vos e consiga manipular dados armazenados. O grande objetivo é dar ao usuário uma visão abstrata dos dados, isso é, o SGBD deve omitir do usuário como ele armazena e manipula as estruturas de dados mantendo-se eficiente. Esse conceito tem direcionado a evolução dos SGBD que buscam formas de manipular as estruturas de dados com eficiência. A busca por simplificar a interação com usuário e abstrair os conceitos das estruturas de dados complexas utilizadas internamente em um SGBD, também são fontes que impulsionam a evolução dos SGBD. Assim, podemos definir abstração de dados como omitir certos detalhes nos concentrando em coisas que consideramos essenciais. Com nossa capacidade mental, podemos selecionar várias características e excluir outras. Abstração é uma visão sem conceituação técnica de uma realidade do mundo que construímos em nossa mente.

O analista na primeira fase de seu trabalho, deve se concentrar nos objetos importantes em um contexto sem se preocupar com o processo de automatização. Esse primeiro modelo chamamos de minimundo. Veja um exemplo na figura 3.



**Figura 3:** Exemplo de Abstração

Ao olharmos para a figura temos em nossa mente o resultado de uma abstração. Em sua mente você identificou a figura como um carro, mesmo que a mesma não apresente detalhes como bancos, volante, maçanetas, retrovisores entre outros. É comum uma abstração ter um nome. Os humanos têm a capacidade de criar representações genéricas da realidade ou modelos conceituais.

O analista deve identificar os elementos geradores de um objeto, sobre quais regras ele opera e quais suas funções. A preocupação é retratar a necessidade de informação que os usuários precisam em um determinado contexto. Para conseguir fazer esse modelo será necessário capturar as regras de negócio, ou seja, como as informações existentes estão distribuídas nos processos do negócio.

## MINIMUNDO

Para o sucesso de uma empresa, é essencial ter uma boa administração com planejamento, organização e controle. É importante que todos saibam o que se pretende alcançar. O analista ao observar parte da empresa, observa o minimundo ou parte do mundo. O minimundo é parte de uma realidade observada pelo analista. O analista deve avaliar o negócio com interesse em como está organizado, como é administrado e quais regras devem ser representadas por controlar o negócio.



### palavra de autor

"Porção da realidade, captada pelo analista, que a gestão de negócios de uma organização tem interesse em observar, controlar e administrar. A complexidade existente no momento de analisar um minimundo pode levar o analista a subdividi-lo em partes menores, às quais damos o nome de visão de processo de negócio." MACHADO (2014 p. 18).

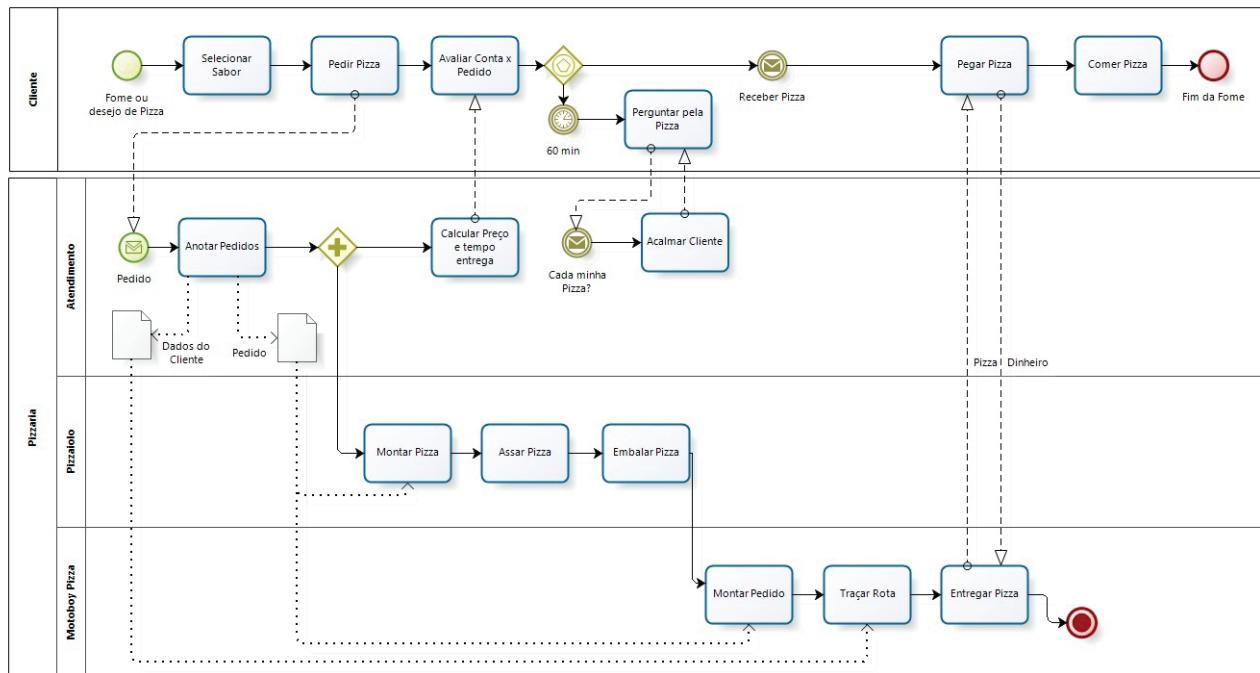
O minimundo apresentará regras que devem aderir ao negócio e o analista deve saber que ocorrerá mudanças durante o desenvolvimento. Às vezes, o analista divide o minimundo em porções menores ao observar o grau de complexidade. A figura 4 apresenta um minimundo modelado com o *Business Process Modeling Notation* (**BPMN**).

As empresas atualmente têm na gestão de negócios o foco na melhoria dos resultados por meio da melhoria dos processos. Para isso, elas se apoiam em Tecnologia da Informação (TI) para automatizar suas atividades. A ideia é criar estruturas conceituais identificáveis em termos das suas atividades e atribuições. Vamos pensar, como exemplo, em uma pizzaria cuja função do negócio é produzir e entregar pizzas a clientes em suas casas. O processo de negócio é um conjunto de atividade ou tarefa que produz um produto ou serviço. O objetivo de um processo de negócio é determinar como o trabalho será produzido e quais ações estão relacionadas entre si. O diagrama apresentado na figura 4 é um exemplo de um processo de negócios, sendo o atendimento, entrega e montagem da pizza exemplos de funções de negócio. As funções no diagrama são horizontais enquanto o processo percorre livremente horizontalmente e verticalmente de forma indistinta entre

### BPMN

Business Process Modeling Notation ou Notação de Modelagem de Processos de Negócios

as funções de negócios. O diagrama BPMN é uma notação visual do minimundo para representar o fluxo do processo que pode ser mapeado para diversos formatos. A modelagem do processo utilizando ferramentas de desenho padronizada facilita o entendimento na própria organização de seus processos e na análise de dados quais são relevantes e precisam ser armazenados.



**Figura 4: Minimundo modelado com o diagrama BPMN**

Utilizamos o diagrama para detalhar com bastante proximidade as complexidades de ambientes reais. A modelagem de processos de negócios é usada para comunicar uma grande variedade de informações.



leitura indicada

Análise e modelagem de processos de negócio: foco na notação BPMN (*Business Process Modeling Notation*) / Rogerio Valle, Saulo Barbará de Oliveira, organizadores. – 1. ed. – 6. Reimpres-são. – São Paulo: Atlas, 2013. O capítulo 7 vai lhe ajudar a entender melhor o BPMN a partir da página 77. Este livro está disponível na Biblioteca Virtual da FAM em Minha Biblioteca.

O BPMN é dividido em três tipos básicos de submodelos: privado, abstrato e de colaboração. Os processos de negócios privados correspondem àqueles que ocorrem dentro da organização. São aquelas atividades que ocorrem internamente. Um processo de negócio privado representa uma parte do processo sem se preocupar com o todo. Os processos de negócios abstratos representam os fluxos e as interações pertencentes a um processo privado com outra entidade de negócio externa. Os processos colaborativos modelam as interações entre dois ou mais processos. O diagrama BPMN apresenta quatro categorias de elementos: objetos de fluxo, objetos de conexão, swimlanes e artefatos.

O evento é algo que acontece durante o processo do negócio. São utilizados para indicar um evento de início, um evento intermediário e um evento de término. Os eventos são representados graficamente por um círculo. O evento de início é verde, o de intermediário é amarelo e o final é vermelho. Também existe uma indicação na linha do círculo, o de início à linha é simples, enquanto que o intermediário apresenta borda dupla e o final têm borda mais espessa. Eventos que aguardam fatos ou possuem uma causa são chamados de *catch*. Eventos que produzem fatos e possuem resultados são chamados *throw*. A figura 5 apresenta os três eventos simples de início, intermediário e final básico.



**Figura 5:** Eventos simples de Início, Intermediário e Fim

Todo processo ou subprocesso deve ter um evento de início indicando que neste ponto um processo é criado. A causa ou resultado de um evento é chamado *trigger* ou gatilho. Os gatilhos são sinalizados dentro dos eventos. Os tipos de gatilhos variam de acordo com cada tipo de evento. Os eventos de inícios podem ser eventos simples, condicional, de tempo, paralelo, uma mensagem, múltiplo ou sinal. A figura 6 apresenta os possíveis usos de um evento de início. Um evento de início sempre será do tipo *catch* pois deve aguardar a ocorrência para iniciar o disparo.



**Figura 6:** Variações do evento de início

O evento de início sem gatilho é disparado sem a definição de um fato específico por isso não possui um gatilho dentro dele. O evento de início de tempo é disparado por um fato temporal como a chegada de uma determinada data ou hora. O evento de início mensagem é disparado quando chega um e-mail, telefone, um documento etc. O evento de início condicional é disparado quando uma determinada condição se torna verdadeira.

Os eventos intermediários são utilizados para indicar que algo ocorreu depois do início e antes do fim do processo. Eventos intermediários podem ser tanto do tipo *catch* como *throw*, ou seja, podem aguardar um fato ou gerar um fato. Em geral são conectados por meio de conectores de fluxo, mas em alguns casos podem ser colocados na borda de uma atividade especificando que irão ocorrer no decorrer de uma atividade. A figura 7 apresenta os tipos possíveis.



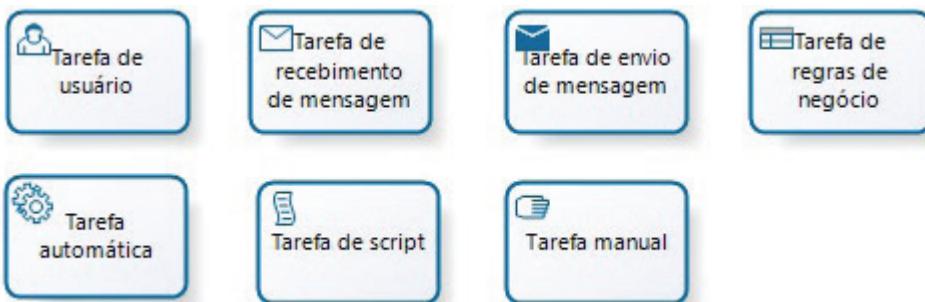
**Figura 7:** Variações do evento intermediário

O evento de término ou fim indica o fim de um processo, sempre será do tipo *throw*. Ele marca que o processo chegou ao fim com um fato. Todo processo deve ter ao menos um evento de término, sendo possível existir inúmeros términos. A figura 8 apresenta as variações do evento de término.



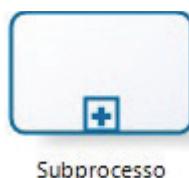
**Figura 8:** Variações do evento de término

A atividade ou tarefa é a ação de um processo que precisa ser realizada. Ela deve indicar o que fazer, sendo identificada por um verbo no infinitivo. É representada por um retângulo com bordas arredondadas contendo um texto dentro da área do retângulo. São exemplos de atividades: Calcular valor da venda, Elaborar roteiro de entrega, Montar Pizza etc. Existem alguns símbolos que podem ser utilizados para representar visualmente a função da tarefa.



**Figura 9:** Variações dos tipos de atividades ou tarefas

A representação gráfica no canto da tarefa ajuda um usuário a identificar o que a tarefa faz e quem a realiza. As atividades podem ser atômicas ou não atômicas. As atômicas são denominadas tarefas e as não atômicas são denominadas de subprocessos. O subprocesso é composto de uma série de outras atividades e fluxos. Quando contraído um subprocesso mostrar apenas um elemento, sendo que existe diversos elementos ocultos, que quando expandidos são exibidos seus elementos. Na versão expandida, apresenta graficamente todos seus detalhes com todos os seus elementos. A figura 10 apresenta a representação de um subprocesso.



Subprocesso

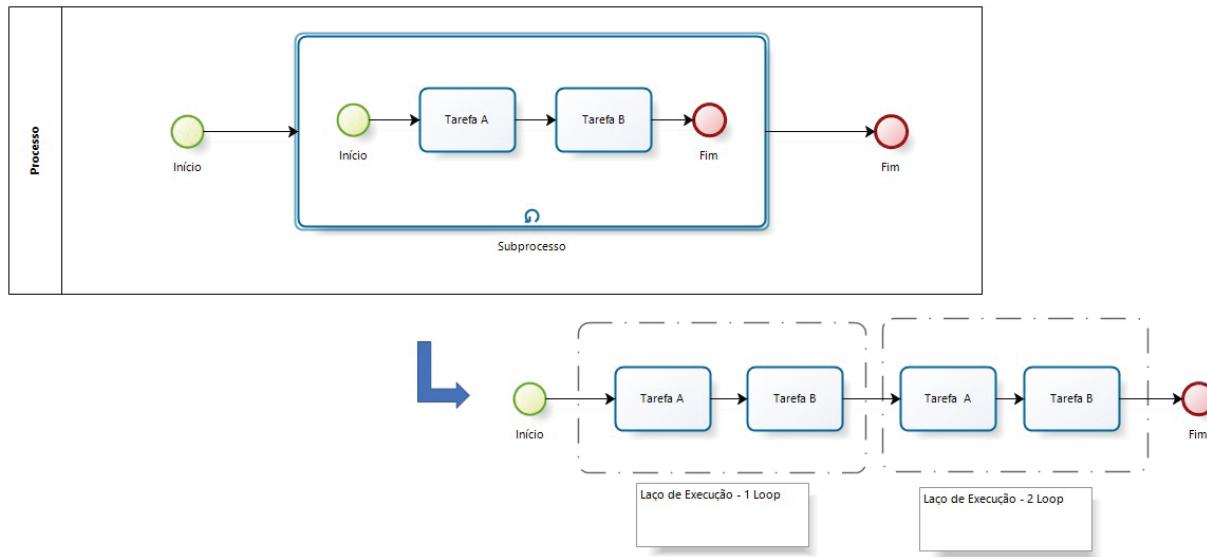
**Figura 10:** Representação gráfica de um subprocesso

Além da Execução normal, tarefas e subprocessos podem ser executados em *loop* ou em paralelo. A figura 11 apresenta as variações entre processos e tarefas.



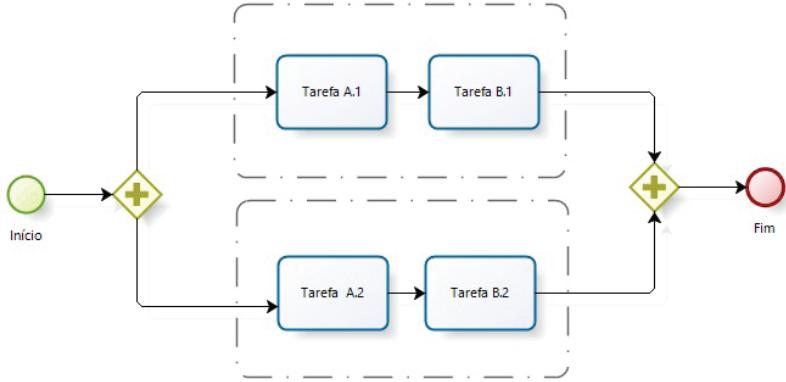
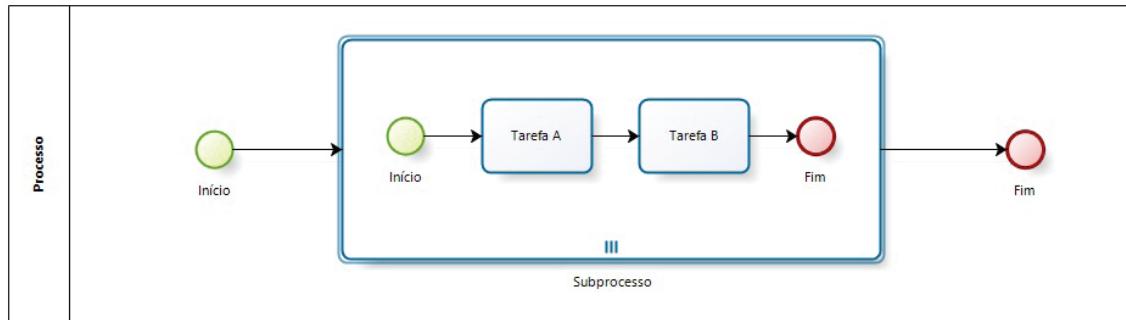
**Figura 11:** Tipos diferentes de tarefas e subprocessos

Utilizamos o *loop* em uma tarefa ou subprocesso toda a vez que precisarmos demonstrar que esta atividade ou processo será executada mais de uma vez. No *loop* você consegue detalhar se o número de repetição será fixo, variável ou condicional. A figura 12 apresenta um subprocesso expandido com duas tarefas A e B que devem ser executadas duas vezes. Note que abaixo do processo representamos apenas didáticamente o que está grafado no processo.



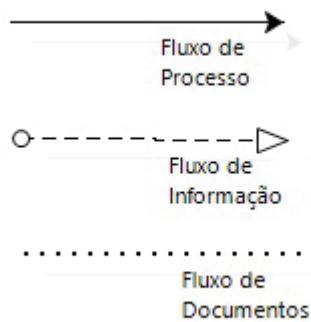
**Figura 12:** Exemplificando o subprocesso *loop*

Você também pode definir uma tarefa ou subprocesso como paralelo. Isso quer dizer que as tarefas criaram instância de si mesma e rodaram em paralelo, sendo que em um determinado instante, haverá um momento de sincronização para voltar ao fluxo normal. A figura 13 apresenta um subprocesso com duas tarefas A e B em paralelo. Abaixo do processo, a fim de tornar melhor o entendimento, exemplificamos o que está representado no processo. Observe que em um instante o fluxo de execução se divide e as tarefas A e B se tornam paralelas tendo múltiplas instâncias. Ao final de suas execuções, existe um momento em que se sincronizam e se tornam um só fluxo normal.



**Figura 13:** Exemplificando o subprocesso paralelo

As atividades são conectadas por meio de conectores de sequência de fluxo, uma linha sólida com uma seta na ponta. Todos os elementos precisam estar conectados um ao outro por uma sequência conforme a ordem em que devem ser realizados. A conexão indica que após a conclusão de uma atividade, a outra poderá iniciar. A figura 14 apresenta os possíveis conectores.



**Figura 14:** Tipos de Conectores

A piscina (*pool*) é o retângulo que representa o processo. Pode ser horizontal ou vertical e normalmente o nome do processo é escrito no

topo do diagrama ou do lado esquerdo. A piscina pode conter apenas um processo de negócio. Uma piscina pode conter inúmeras raias (*lanes*). Elas são utilizadas para representar as funções, departamentos envolvidos ou participantes envolvidos em um processo. As raias são faixas funcionais. Normalmente, representam as funções de um processo (vendas, *marketing*, comercial, fiscalização, etc.) mas também podem representar unidades organizacionais e até cargos. As raias são utilizadas para demonstrar quem é responsável pelas atividades ao longo do processo. Se um processo se comunicar com outro, uma segunda piscina deve ser criada. E em alguns casos, podemos omitir detalhes de uma piscina mostrando apenas que a mesma se comunica com a outra. Neste caso a piscina que não apresenta os detalhes é uma caixa preta (*black box* ou *collapsed*).



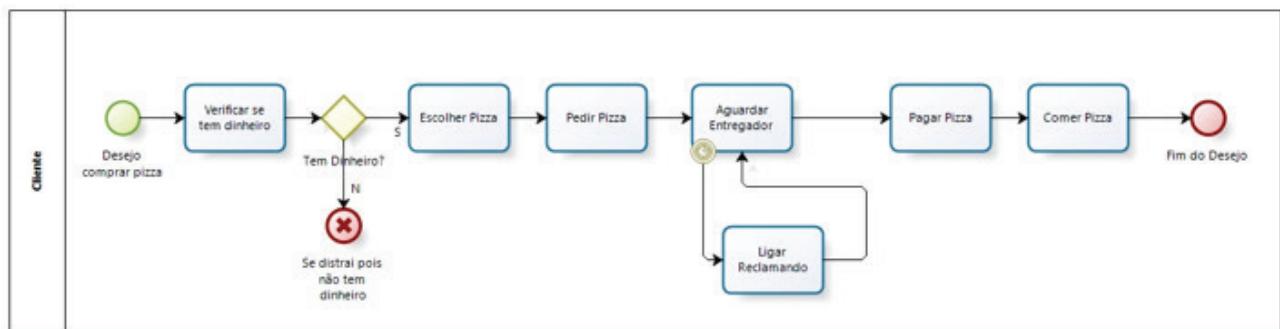
**Figura 15:** Piscina e raias

Objetos de dados são utilizados para representar documentos físicos ou eletrônicos que são gerados ou atualizados ao longo do fluxo. São elementos de anotação que podem ser utilizados para gerar informação complementar ao processo. Representam um conjunto de informações, uma atividade ou troca de mãos. São representados por uma página com a ponta dobrada. O Armazenamento de dados pode ser utilizado para representar um banco de dados ou repositório de documentos.



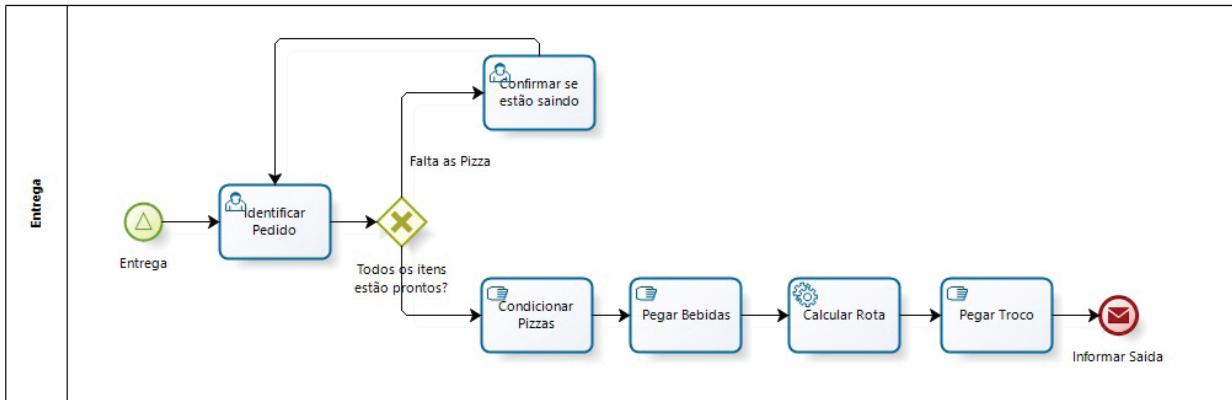
**Figura 16:** Documentos e Banco de Dados

O conector fluxo de documentos é utilizado para conectar os artefatos objetos de documentos e o banco de dados às tarefas que fazem uso deles. Agora que já conhecemos os conectores de fluxo, temos que apresentar o desvio de fluxo ou gateways. Os gateways são elementos utilizados para controlar como um fluxo diverge ao longo da execução do processo. Eles são opcionais e você só deve utilizá-los se for necessário controlar o fluxo de execução. É pelos gateways que você consegue descrever como o fluxo se comporta ao ocorrer uma exceção. O gateway é representado graficamente por um losango. O símbolo interno identifica o que a interpretação representa. A figura 17 apresenta um desvio simples no fluxo de execução.



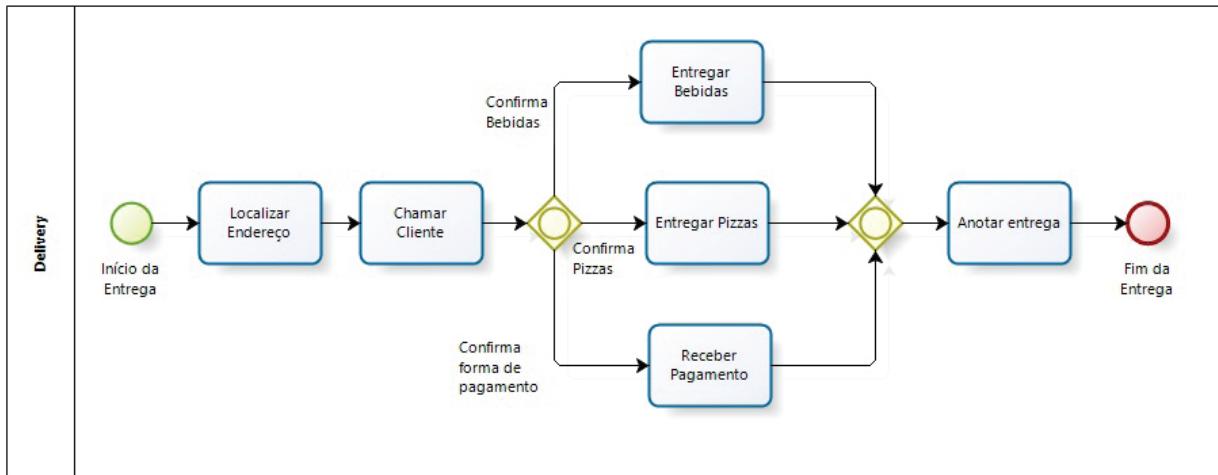
**Figura 17:** Desvio simples – gateway

O gateway exclusivo indica que apenas um dos caminhos criados pode ser seguido. Esse gateway é representado graficamente com um "x" no centro. Os conectores ligados ao gateway podem apresentar uma descrição que ajudem a identificar qual opção se segue por este caminho. Além de separar um fluxo, este gateway pode unificar fluxos. A figura 18 apresenta o desvio exclusivo.



**Figura 18:** Desvio condicional exclusivo

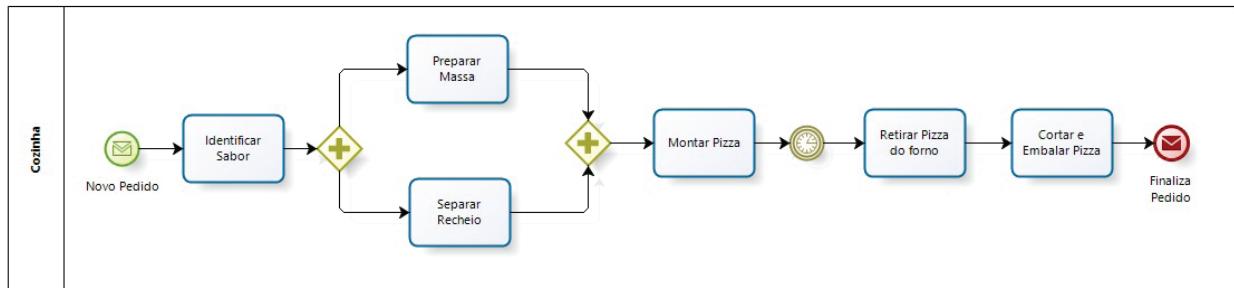
Outro tipo diferente de gateway é o desvio condicional inclusivo, que permite que dois ou mais fluxos sejam seguidos simultaneamente. O processo avalia a condição e se for verdadeira segue a execução. Quando esse gateway for utilizado em um determinado ponto, você deverá utilizá-lo novamente para sincronizar a execução paralela que foi criada. A figura 19 apresenta o desvio condicional inclusivo.



**Figura 19:** Desvio condicional inclusivo

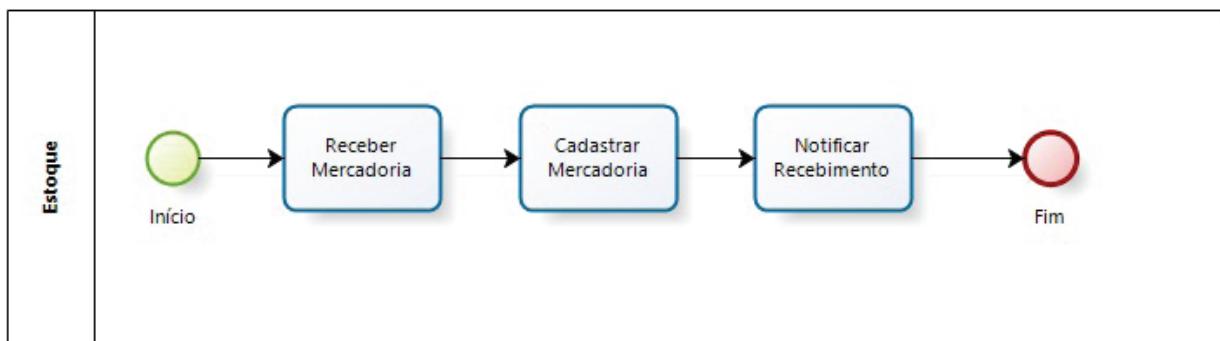
É possível indicar tarefas que ocorrem em paralelo fazendo uso do gateway paralelo, que indica que todas as linhas de execução que saem desse gateway são executadas simultaneamente. Esse gateway tem a representação gráfica de um losango com o símbolo de "+" em seu interior. Quando você cria um fluxo paralelo, se torna necessário utilizá-lo novamente para convergir as execuções em paralelo indicando o momento que as tarefas serão sincronizadas em uma novamente. A

figura 20 apresenta o processo da cozinha de uma pizzaria com duas atividades ocorrendo em paralelo.



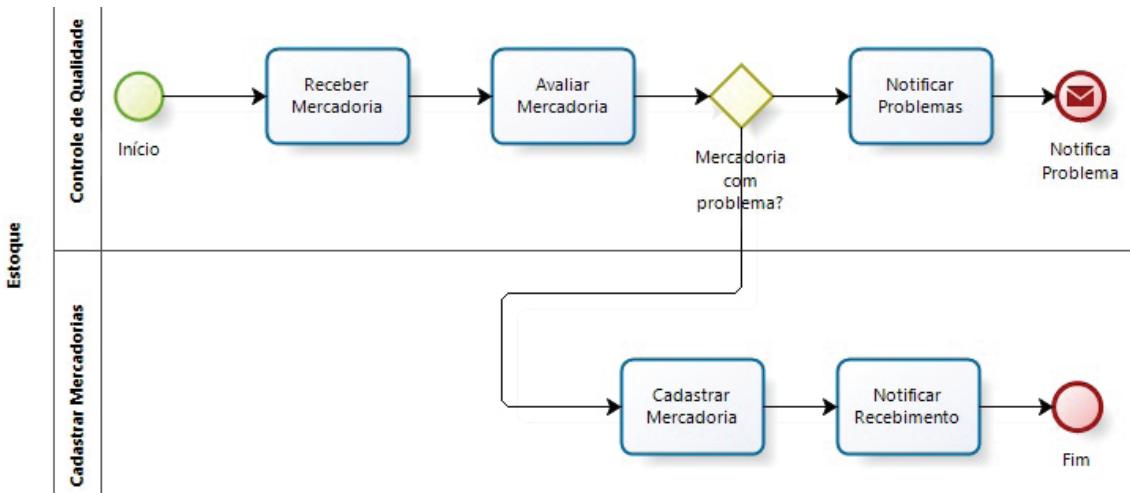
**Figura 20:** Desvio condicional paralelo.

Ao diagramar um processo de negócio, devemos ter em mente que ele inicia simples e que vai evoluindo à medida que refinamos o entendimento do minimundo. A figura 21 apresenta o processo de controle de estoque da pizzaria e a figura 22 apresenta o mesmo sendo refinado.



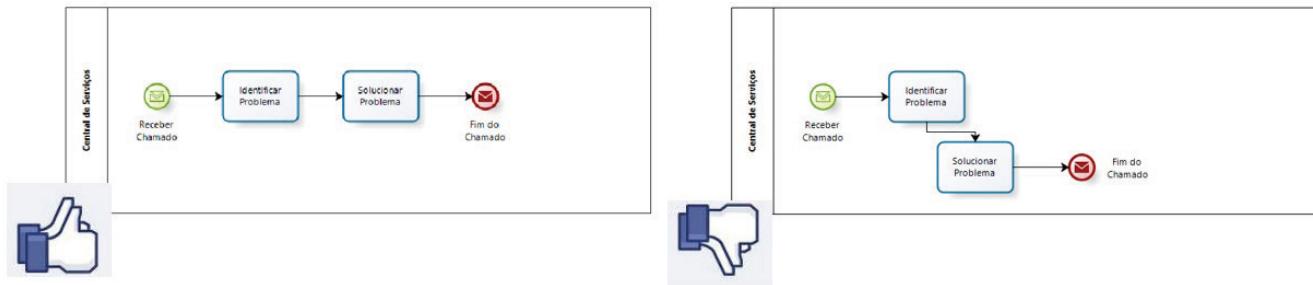
**Figura 21:** Diagrama de Negócio

O processo mapeado pode ser refinado com adição de novos atores, como por exemplo o controle de qualidade que valida se a mercadoria recebida está em condições de ser utilizada na pizzaria. A figura 22 apresenta um refinamento progressivo na visão do negócio da pizza-ria. A modelagem do minimundo e suas regras de negócios vão gerar diversos diagramas BPMN. Os diagramas gerados devem ser apresentados a todos os membros da empresa, para conhecer o processo e modificar ou refinar o diagrama a qualquer momento. Os diagramas podem conter informações sobre o negócio, informações operacionais, informações específicas do processo e informações técnicas.



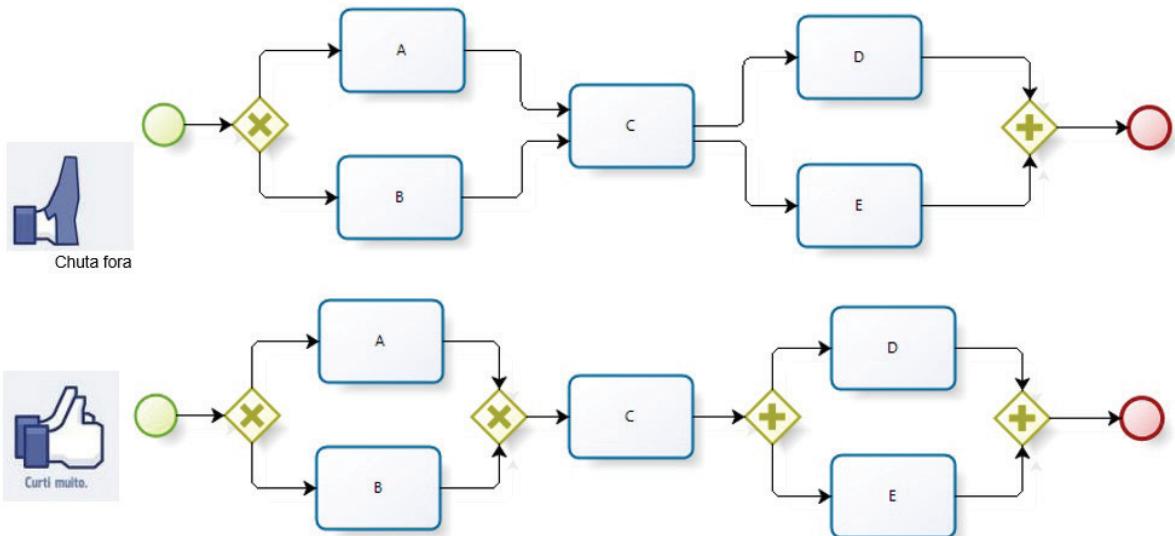
**Figura 22:** Diagrama de Negócio sendo refinado

Um dos erros mais comuns ao criar diagramas BPMN e não manter o alinhamento do processo dentro da mesma raia. A figura 23 apresenta o certo e o errado.



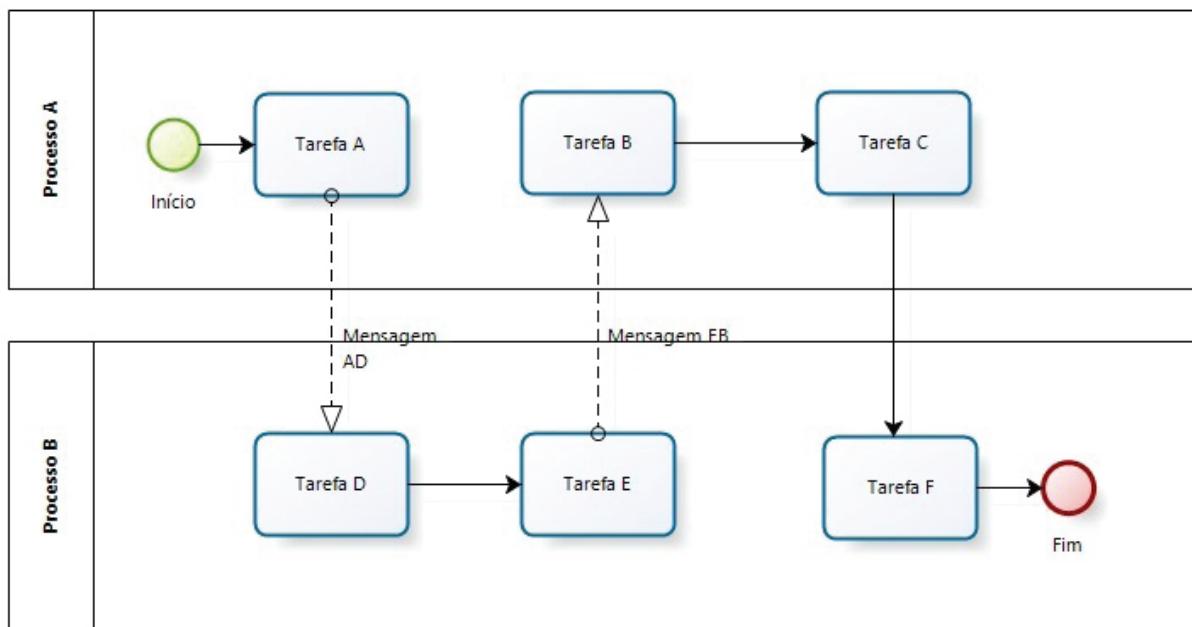
**Figura 23:** Erro na apresentação da raia

Outro erro comum é ao se criar em um diagrama BPMN com um gateway condicional exclusivo ou paralelo, não sincronizar corretamente os fluxos. A figura 24 apresenta o erro e como deve ser feito.



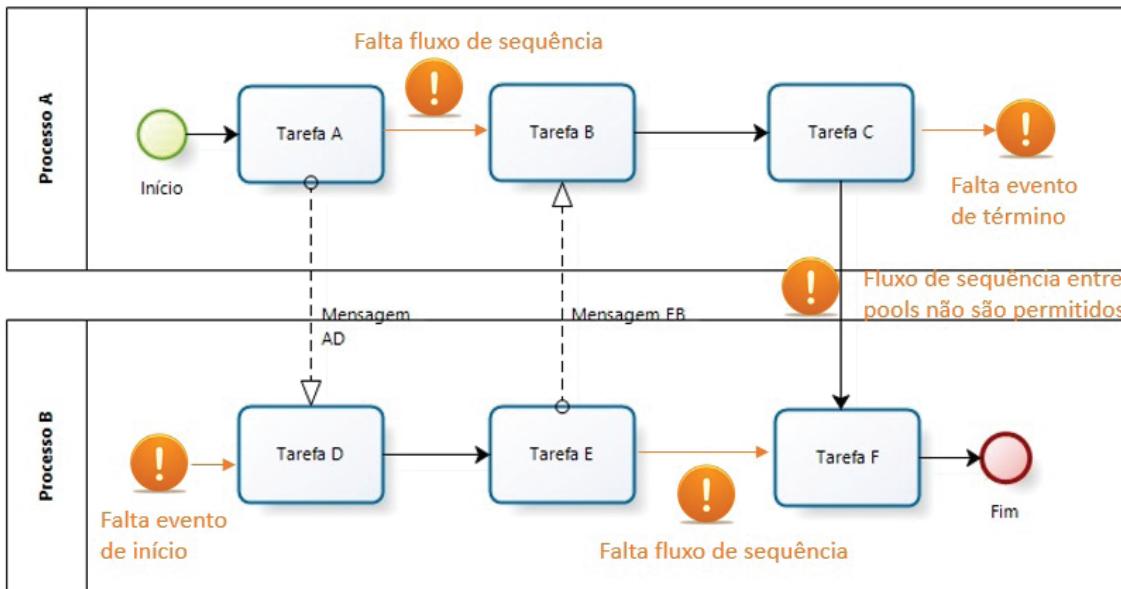
**Figura 24:** Erro ao criar um gateway paralelo

Outros erros que muitos cometem são apresentados na figura 25. Avalie a figura e tente descobrir o que está faltando e o que não está correto.



**Figura 25:** Erros comuns ao se criar um diagrama BPMN.

Agora observe as indicações das correções no diagrama BPMN ilustradas na figura 26.



**Figura 26:** Correções dos erros apresentados na figura 25.

Agora que sabemos como modelar o minimundo e representá-lo graficamente, voltemos nossa atenção ao banco de dados.



dica

Existem diferentes ferramentas para modelagem do BPMN, uma que pode facilitar a modelagem se encontra em <http://www.bizagi.com/en/products/bpm-suite/modeler>. Para você escolher uma ferramenta que mais lhe agrada e ajude a modelar o minimundo, faça uma pesquisa em diferentes fóruns de usuários e note seus comentários sobre as ferramentas utilizadas.

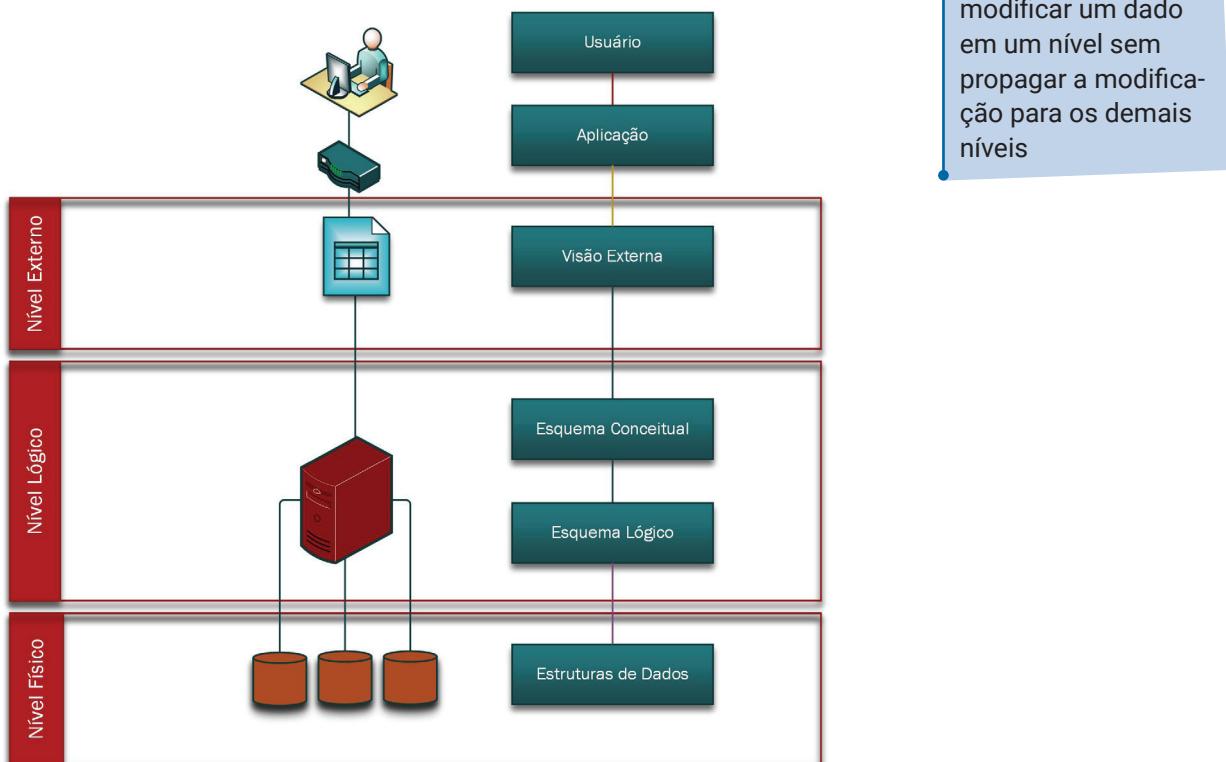
## BANCO DE DADOS

É um conjunto de dados devidamente relacionados de forma a minimizar o uso de memória primária e secundária. É uma coleção ordenada de dados com uma lógica coerente. O banco de dados deve ser projetado, construído e preenchido com um propósito claro e definido. Ele representa o minimundo, assim, uma alteração no minimundo implica alterar o banco de dados.

## SEGURANÇA EM BANCO DE DADOS

Uma das regras de segurança que deve ser fornecida pelo SGBD deve ser permitir que se crie visões de um determinado banco de dados. Um administrador do banco de dados deve ter acesso a detalhes físicos e conceituais dos dados, enquanto um usuário deve ter restrição quanto ao seu nível de visão, lhe sendo negado o acesso às estruturas físicas e inclusive à dados armazenados nessas estruturas. Por exemplo: imagine um determinado contador que precisa acessar a tabela de funcionários, seu acesso deve permitir visualizar apenas alguns campos do registro armazenado na estrutura. Assim, ele pode visualizar o nome do funcionário, sua data de nascimento, se tem filhos, mas não é necessário acessar o seu salário. O nível de visão permite diferenciar entre nível conceitual e físico e especificar os campos que serão visíveis ou inacessíveis.

Conhecer os níveis de visão permite mudar um nível. O ideal é conseguir modificar um nível sem afetar o outro, e quando se consegue isso temos a **independência de dados**. Existem dois tipos de independência de dados: a lógica e a física. A figura 27 apresenta os dois tipos de independência.



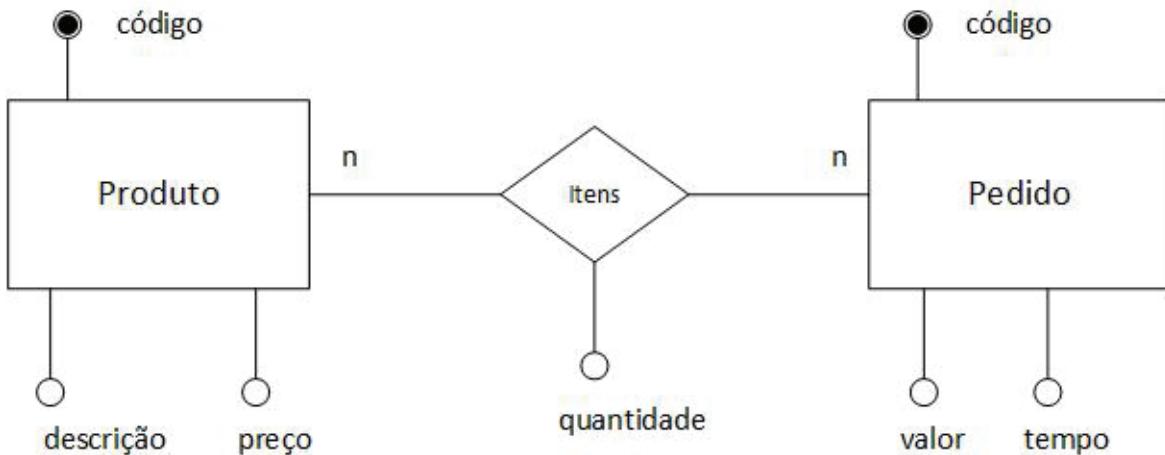
**Figura 27:** Arquitetura de camadas de um SGBD

A independência física é a capacidade de modificar o banco de dados sem a necessidade de reescrever ou modificar a aplicação que faz uso do banco de dados. Muitas vezes, será necessário modificar estruturas de tabelas e comandos de manipulação no banco de dados para otimizar ou um *tuning* de performance e este processo não pode implicar reprogramação das aplicações. A independência lógica é a capacidade de modificar o esquema conceitual sem propagar as modificações para a camada de aplicação. É similar a TAD uma vez que irá esconder do usuário a implementação e permitirá que se concentre na estrutura geral. Vale lembrar, que as modificações conceituais ocorrerão sempre que houver uma modificação física no banco de dados.

Os SGBD devem ser flexíveis ao ponto de serem customizados e aplicados à diferentes aplicações, mantendo-se eficientes e eficazes. Os projetos lógicos e conceituais devem apresentar uma grande capacidade de adaptação e prever um volume de dados armazenados à curto, médio e longo prazo. Ao desenvolver um projeto, o grau de abstração de dados deve ser alto e genérico, possibilitando a utilização de diferentes tipos de SGBD. O projeto deve prever uma interface ágil que facilite a aprendizagem e minimize o esforço para aprender a integrar com o sistema. A portabilidade e interface independente do SGBD são desejáveis. A portabilidade refere-se à capacidade de modificar o Sistema Operacional entre Unix, Windows, Linux etc. Os SGBD sofrem influências de diferentes semânticas no desenvolvimento de suas interfaces, sendo as mais relevantes: procedural, orientada a objeto, orientado a evento e relacional.

## MODELO CONCEITUAL

Apresenta os dados sobre uma visão global de um problema. Mostra como os dados se relacionam independente da tecnologia e da implementação. O modelo conceitual deve registrar quais dados deverão existir no banco de dados sem descrever como serão armazenados no SGBD. Esta deve ser a primeira etapa de um projeto de banco de dados. O modelo conceitual usualmente utiliza o Diagrama Entidade-Relacionamento (DER).



**Figura 28:** Modelo Conceitual representado com DER

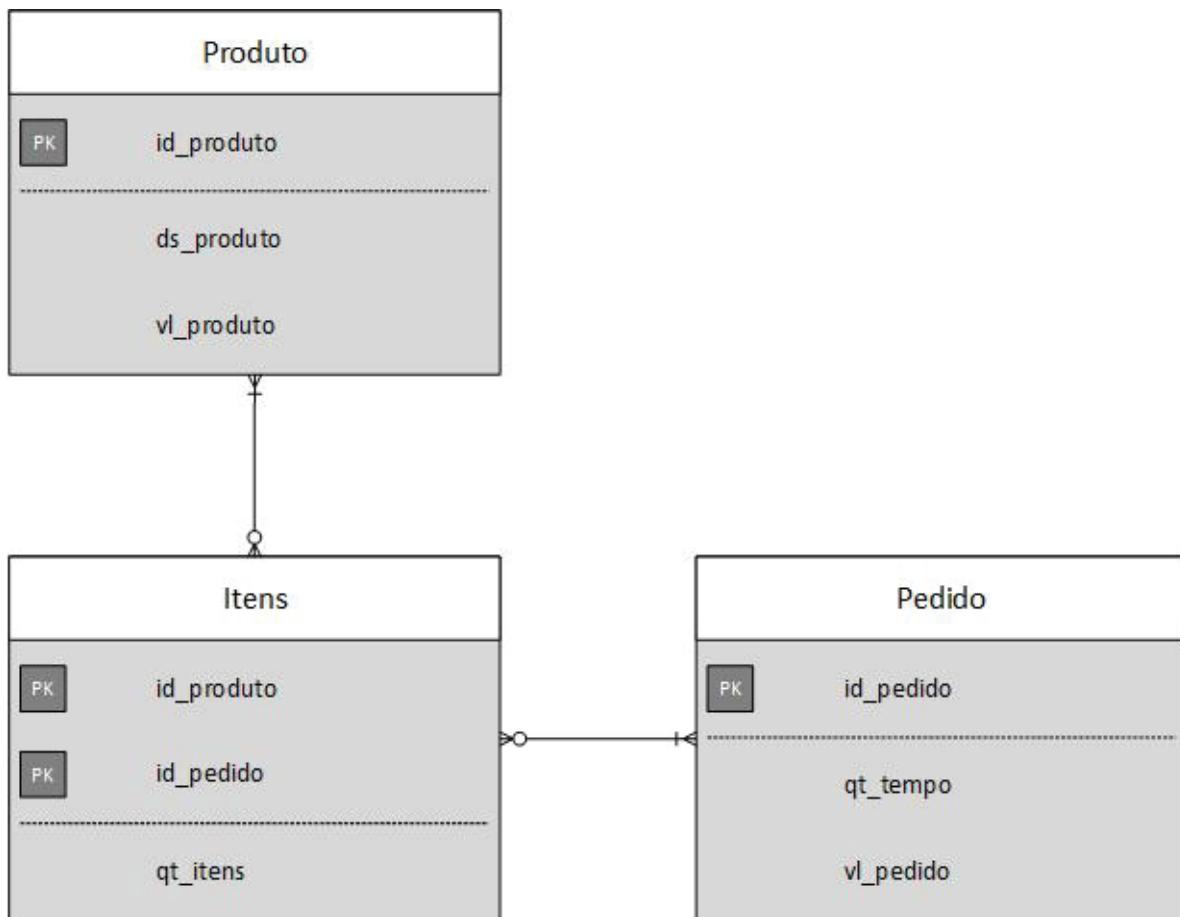
Esse modelo informa que existirá uma tabela com a descrição, o código e o preço dos produtos da empresa. Haverá também uma tabela com as informações dos pedidos, como o valor, o tempo de entrega e o código. Por fim, o diagrama informa que existe um relacionamento entre produto e pedido. Esse relacionamento também deve gerar uma tabela com a quantidade, código produto e código\_pedido. Vamos aprender a como modelar o DER e o que ele representa em outra aula. É importante destacar que o modelo conceitual não retrata o uso, formas de acesso ou estruturas de dados. Ele nunca deve ser construído com considerações sobre o processo do negócio e as operações e sim sobre os dados do negócio.

## MODELO LÓGICO

Tem início após a criação do modelo conceitual. Ele considera as possíveis tecnologias existentes no SGBD. É uma abstração da implementação ou modelo de SGBD, sendo o modelo independente do tipo e da distribuição do SGBD. O modelo lógico representa a estrutura de dados conforme visto pelo usuário. Detalhes que influenciam o desempenho, programação interna do SGBD e estruturas de acesso à informação não são representados no modelo lógico e sim no modelo físico do sistema. Por não serem padronizados, os modelos físicos variam de SGBD para SGBD sendo apenas utilizado para otimização. Pular a fase de elaboração do modelo conceitual, leva o analista a distorcer a realidade do minimundo por adotar as restrições tecnológicas em sua

abordagem inicial. Você deve evitar cair na armadilha de pular o modelo conceitual para que seus modelos tenham aderência ao minimundo, minimizando os erros de interpretação.

O modelo lógico vai apresentar as estruturas que estarão no banco de dados permitindo sua portabilidade, ou seja, que você o implemente em qualquer SGBD. A Figura 29 apresenta o modelo lógico da pizzaria retratando apenas a parte apresentada no modelo conceitual.



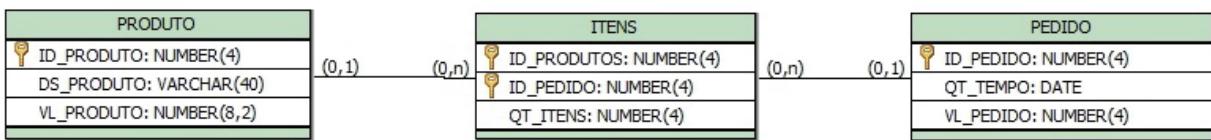
**Figura 29: Modelo Lógico**

O modelo deve definir quais as tabelas que serão criadas e os nomes das colunas. O modelo lógico apresentado no formato texto ficaria assim:

```
Produto(id_produto, ds_produto, vl_produto)
Itens(id_produto, id_pedido, qt_itens)
Pedido(id_pedido, qt_tempo, vl_pedido)
```

## MODELO FÍSICO

É construído após a conclusão da elaboração do modelo lógico. Apresenta as estruturas de armazenamento como: tipo, tamanho dos campos, índices, nomenclaturas, obrigatoriedade, gatilhos etc. Visa economizar os recursos computacionais e deve ser projetado levando em conta os requisitos de processamento. Esse modelo detalha o modelo lógico e será a etapa final do projeto do banco de dados. Para tanto, será utilizada a Linguagem de Definição de Dados (DDL). Você deve criar um *script* com os comandos de criação do banco de dados que serão executados pelo SGBD.



**Figura 30:** Modelo Físico da Pizzaria

O modelo do *script* gerado em DDL escrito em SQL para criar essas tabelas fica assim:

```
CREATE TABLE PRODUTO (
    ID_PRODUTO NUMBER(4) PRIMARY KEY,
    DS_PRODUTO VARCHAR(40) NOT NULL,
    VL_PRODUTO NUMBER(8,2) NOT NULL
);

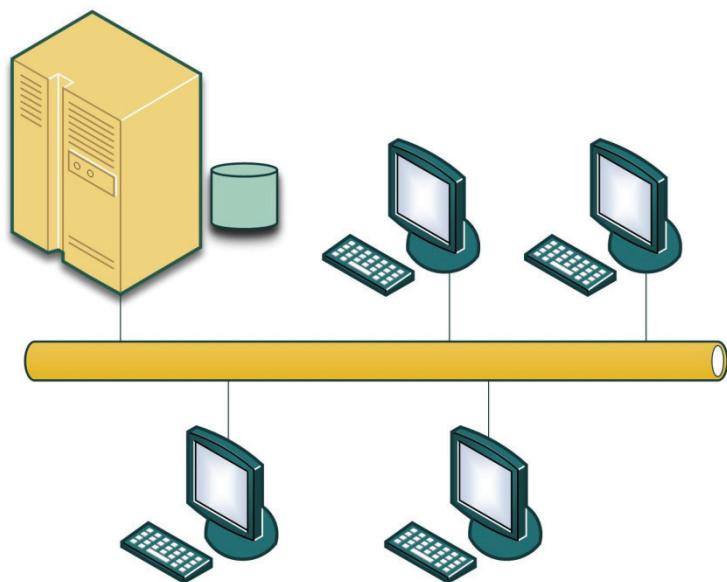
CREATE TABLE ITENS (
    ID_PRODUTOS NUMBER(4),
    ID_PEDIDO NUMBER(4),
    QT_ITENS NUMBER(4) NOT NULL,
    PRIMARY KEY(ID_PRODUTOS, ID_PEDIDO),
    CONSTRAINT ITENS_ID_PRODUTOS_FK FOREIGN KEY (ID_PRODUTOS)
        REFERENCES PRODUTO (ID_PRODUTO),
    CONSTRAINT ITENS_ID_PEDIDO_FK FOREIGN KEY (ID_PEDIDO)
        REFERENCES PEDIDO (ID_PEDIDO)
);

CREATE TABLE PEDIDO (
    ID_PEDIDO NUMBER(4) PRIMARY KEY,
    QT_TEMPO DATE NOT NULL,
    VL_PEDIDO NUMBER(4) NOT NULL
);
```

## ARQUITETURAS

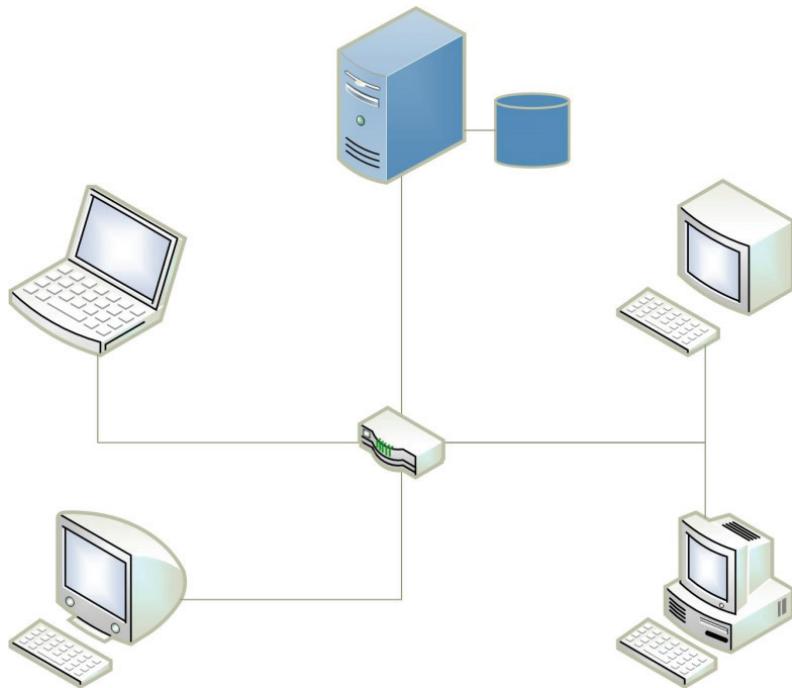
Os *mainframes* formam as primeiras arquiteturas utilizadas para processar todas as funções dos sistemas, sendo suas responsabilida-

des processar: os programas, interfaces e funcionalidades de banco de dados. Os usuários acessavam os *mainframes* por meio de terminais que não tinham poder de processamento, limitando-se nestes terminais apenas à visualização. Todo o processamento era feito remotamente no *mainframe*, sendo enviado para o terminal apenas as informações que seriam visualizadas. A comunicação na rede era intensa e o custo dos *mainframes* era elevado. A figura 31 apresenta a Arquitetura de *Mainframe*.



**Figura 31:** Arquitetura de Mainframe

Com o surgimento dos computadores pessoais (PC), com preços bastante acessíveis, muitas pequenas empresas começaram a utilizar essas máquinas e cresceram tendo elas como equipamentos que as auxiliaram em tarefas administrativas. As primeiras redes se inspiravam na arquitetura dos *mainframes* de forma que um PC se tornava um terminal e o outro se tornava o servidor que centralizava toda a funcionalidade, tendo de processar os programas aplicativos, dados e interface do usuário. Com o tempo, notou-se que por existir poder de processamento em ambas as máquinas, algumas tarefas poderiam ser alocadas ao servidor e outras ao cliente. Assim surgiu a arquitetura cliente-servidor onde o processamento é dividido, cabendo ao servidor prestar alguns serviços, e ao cliente processar algumas tarefas, explorando o processamento disponível no cliente. A figura 32 apresenta a Arquitetura Cliente-Servidor.



**Figura 32:** Arquitetura Cliente-Servidor

A arquitetura cliente-servidor utiliza a estrutura fornecida pela rede para dividir em diferentes máquinas as tarefas, criando serviços especializados. Neste ambiente de computação, uma grande porção das máquinas serão estações de trabalho e algumas máquinas serão servidores. Teremos servidores especializados como: servidor de arquivos que mantém os arquivos de diferentes clientes armazenados e seguros, o servidor de impressão que podem estar conectados a várias impressoras e presta serviço a inúmeros clientes, controlando as impressões.

Todas as máquinas clientes devem disponibilizar para o usuário as interfaces apropriadas para utilizar esses servidores. Também cabe à máquina cliente utilizar seu poder de processamento para executar aplicações locais. Esta arquitetura é muito popular, por ser de fácil implementação, dada à clara separação das funcionalidades que rodam nos clientes e nos servidores, por utilizar melhor o servidor ao delegar as tarefas mais simples às máquinas clientes mais baratas e por permitir ao usuário executar uma interface gráfica que lhe é familiar, ao invés de usar a interface do servidor. A arquitetura cliente-servidor foi incorporada por diferentes SGBDs comerciais, sendo que a mais adotada foi os Sistemas de Gerenciamento de Banco de Dados Relacionais (SGBDR) no lado do servidor.

O SGBDR trabalha com consultas (*Queries*) no servidor de consultas ou servidor de transação. Esse serviço que roda no servidor implementa um interpretador de comandos SQL que serve como interface com o cliente. Cada cliente deve enviar suas consultas e cabe ao servidor interpretar tais consultas e respondê-las de forma adequada. O cliente pode também criar um dicionário de dados com informações sobre a distribuição dos dados, números de consultas locais que podem ser executadas e definir como uma consulta global pode ser decomposta.

#### **Back-end machine**

servidor da arquitetura client-server

#### **Front-end machine**

cliente da arquitetura client-server

É comum no mercado chamar o servidor de **back-end machine** e o cliente de **front-end machine**. O SQL ao se tornar a linguagem padrão dos SGBDR acabou criando um ponto de divisão entre a lógica do servidor e do cliente.

Atualmente, as tendências das arquiteturas se dividem entre as plataformas: centralizada, sistemas de computadores pessoais, cliente-servidor e distribuídas.

Na plataforma centralizada, existe um computador de grande poder de processamento que hospeda o SGBD e processa as várias requisições das aplicações. A vantagem dessa arquitetura é que ela suporta que um grande número de usuários manipule um grande volume de dados. A desvantagem é o alto custo nas soluções centralizadas e na aquisição de *mainframes*.

Os computadores pessoais fazem seus processamentos sozinhos ou *stand-alone* o que significa que rodam como servidor e como clientes. Sua principal vantagem é a arquitetura simplificada que chamamos de padrão Xbase. Como desvantagem, podemos listar inconstância de performance pois a velocidade irá depender das tarefas alocadas no *hardware*.

Na arquitetura cliente-servidor, temos uma divisão de tarefas entre ambas as máquinas, ficando o *front\_end* (cliente) responsável por executar tarefas do aplicativo, como interface (tela gráfica), processamento de entrada (validação de campos) e tratamento de saída. O *back\_end* ou servidor executará as consultas ao Banco de Dados e retornará os resultados ao cliente. É uma arquitetura popular, apesar de sua sofisticação de implementação, à medida que implementa tratamento de transações, tais como *commits*, *rollbacks*, *store procedures* e *triggers* que você verá posteriormente. As principais

vantagens são a divisão do processamento e redução do tráfego de dados na rede.

Na arquitetura distribuída, a informação está dividida em diversos servidores que atuam como cliente-servidor. As consultas de uma aplicação são feitas a qualquer um dos servidores de forma indistinta, e este deve conseguir obter os dados, sem informar o processo realizado para isso. Assim, o usuário tem a impressão que a rede responde de forma única, pois ele não conhece seus servidores e nem suas redundâncias e conflitos. A característica básica é a existência de diversos programas consultando a rede de dados sem conhecer quais servidores dispõem desses dados. As vantagens são a flexibilidade e desempenho que podem ser atingidos. Como desvantagens aparecem a elevação na complexidade do desenvolvimento e o equilíbrio entre segurança e performance.

## DICIONÁRIO DE DADOS

O dicionário de dados é um conjunto de elementos que definem e representam um dado. Dentro do SGBD é um conjunto de tabelas em que apenas podemos ler e consultar e que mantêm as seguintes informações:

- definição dos elementos de dados;
- perfil dos usuários, papéis e privilégios;
- descrição dos objetos;
- *stored procedure* e *triggers*;
- restrições de integridade;
- estruturas gerais das bases;
- informação de verificações;
- alocação de espaço.

O maior benefício de se ter um dicionário de dados bem definido é a consistência entre itens de dados armazenados em diferentes tabelas. Imagine que você armazene, em diferentes tabelas, informações sobre números de telefone. Por utilizar um dicionário de dados, o formato número de telefone poderá ser definido com uma máscara "(99) 9 9999-9999" e deverá ser obedecido por todas as tabelas que utilizam tal dado.

Nesta segunda aula, você foi apresentado à importância do banco de dados e conheceu os principais termos que utilizaremos nas próximas aulas. Você tomou conhecimento da evolução histórica dos bancos de dados e aprendeu como as pesquisas conduziram para o atual modelo. Tivemos uma noção de modelagem do minimundo com o diagrama BPMN e vimos os conceitos iniciais do modelo conceitual, modelo lógico e modelo físico. Esses três últimos serão detalhados nas próximas aulas. Por fim, você pode ver as arquiteturas utilizadas com suas vantagens e desvantagens.

## Atividades

- 01.** Quais as vantagens e desvantagens entre um sistema de processamento de arquivos e um sistema de gerenciamento de banco de dados?

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- 02.** Quais os tipos de arquiteturas de banco de dados, vantagens e desvantagens de cada uma?

---

---

---

---

---

---

---

- 03.** A partir da solicitação da Dra. Vânia, modele em BPMN o processo financeiro da Clínica Médica. A Dra. Vânia deseja um sistema que permita controlar o faturamento de sua clínica médica. Ela deseja obter no fim do mês, um relatório com o valor recebido pelas consultas particulares e o faturado pelos convénios. Cada convênio paga um valor diferente por consulta e credita em prazos diferentes (alguns creditam em 45 dias, outros em 60 etc.) a partir da data de apresentação da consulta ao plano. A previsão de pagamento deve aparecer no relatório. O valor da consulta particular é fixo, mas um ou outro paciente pode receber um desconto. Sendo assim, o valor efetivo pago deve ser registrado.

---

---

---

---

---

---

---

## Anotações

# AULA 3

## MODELAGEM DE DADOS

### NESTA AULA

- » Etapas de um projeto
- » Modelagem do E-R
- » Atributos e chaves
- » Cardinalidade
- » Especialização ou Generalização
- » Condicionalidade de Relacionamento
- » Relacionamentos Contingentes
- » Agregação
- » Validação de Diagramas
- » Aspectos Temporal
- » Modelo Lógico
- » Notação da Engenharia da Informação
- » Regras de integridade.

### ■ METAS DE COMPREENSÃO

- » Conhecer as etapas de um projeto de banco de dados.
- » Compreender o Modelo Entidade-Relacionamento (MER)
- » Aprender a criar Diagramas Entidade-Relacionamento (DER) incorporando os conceitos de integridade, chave e cardinalidade.

### ■ APRESENTAÇÃO

O objetivo nesta aula é tratar das fases de projeto e modelagem de um Banco de Dados Relacional. Você deve entender as fases envolvidas e os conceitos do Modelo Entidade-Relacionamento (MER). O estudo de modelagem de banco de dados o ajudará a criar um conceito abstrato de banco de dados que poderá ser implementado em qualquer SGBD. Também você notará que existem alguns desafios em um projeto de Banco de Dados.

Para entender estes conceitos, esta unidade está organizada da seguinte forma:

- » a seção 2 apresenta as etapas de um projeto banco de dados;
- » a seção 3 mostra a modelagem entidade-relacionamento;
- » a seção 4 apresenta os conceitos de chaves e atributos;
- » a seção 5 apresentar cardinalidade e DER;
- » a seção 6 introduzir conceitos de agregação, generalização, condicionalidade e contingentes;
- » a seção 7 ensina a validar diagramas e convertê-los para modelo lógico;
- » a seção 8 introduz as regras de integridade.

Não deixe de utilizar as bibliografias associadas à unidade. Bom estudo!

# ■ ETAPAS DE UM PROJETO

## Levantamento de Requisitos

entender o que será desenvolvido e quais os dados serão processados pelo sistema.

## Análise de Requisitos

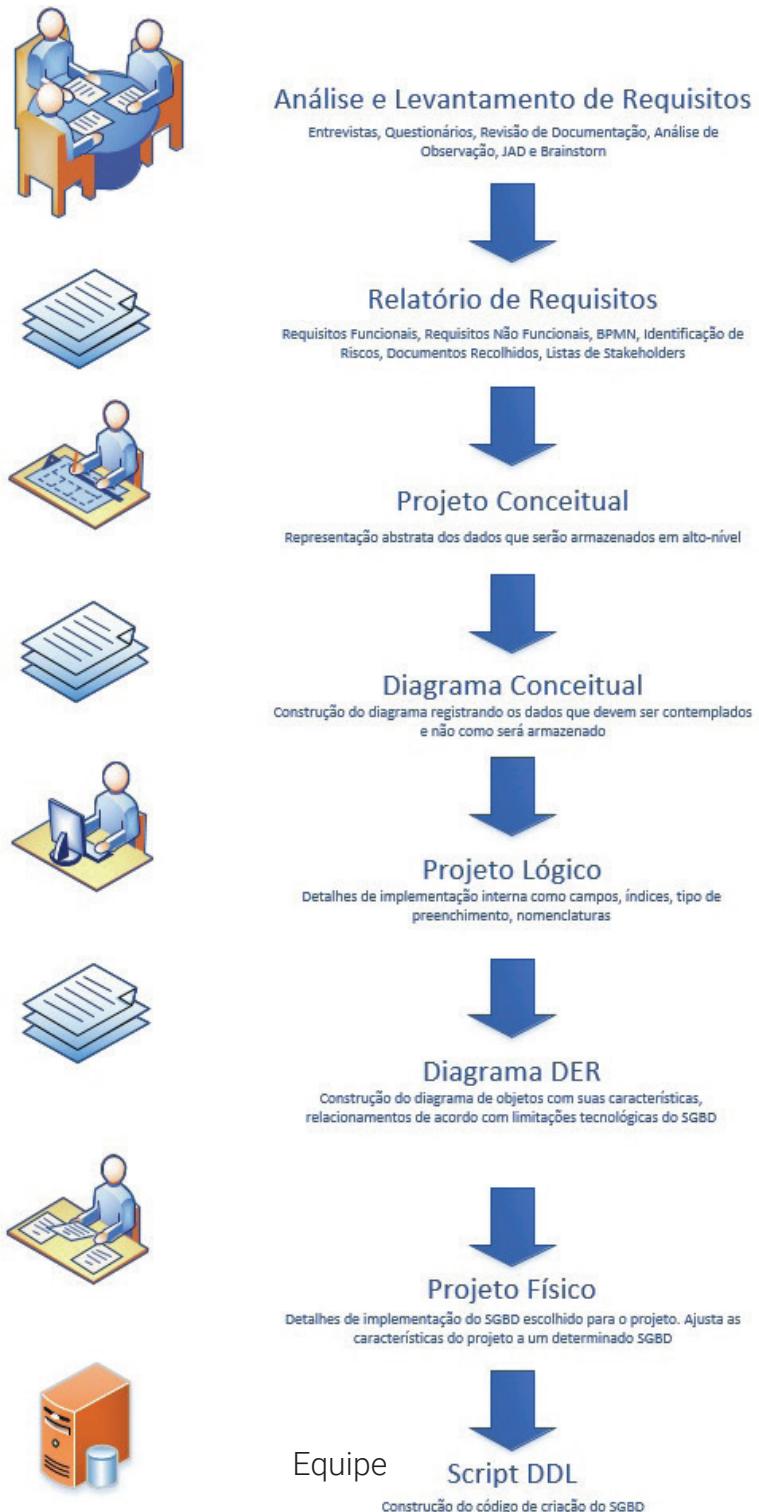
descoberta, modelagem e especificação dos requisitos do sistema.

## Relacionamento

uma associação entre objetos caracterizado por um verbo.

A técnica de projeto mais utilizada é iniciar por fazer um **levantamento de requisitos**. Esta fase tem como objetivo entender os requisitos do software que será desenvolvido e quais os dados serão processados pelo sistema. É nesta etapa que se modela e entende o negócio da empresa, suas necessidades e funcionamento. O processo de levantamento de requisitos se dá por técnicas de entrevistas com usuários. Quanto mais se entende os requisitos de software, maior a probabilidade de sucesso. Um software pode ser bem codificado e funcional, mas se não atender as necessidades dos usuários, acabará por não ser utilizado. Por isso, a **análise de requisitos** é um momento de descoberta, modelagem e especificação que deve ocorrer em vários momentos, e que podem ser refeitos. Pode parecer uma tarefa simples, mas devido ao alto grau de complexidade e troca de informações, se mostra um grande desafio. Vão ocorrer, nesta fase, interpretações errôneas, informações falsas, ambiguidades, conflitos e mudanças durante o processo de levantamento de requisitos. A segunda etapa é elaborar o projeto conceitual, que tem como base os relatórios criados na fase de análise e levantamento de requisitos. Nesta fase, inicia-se a modelagem do projeto conceitual. O modelo foi criado por Peter Chen em 1976 que apresenta o mundo real como objetos chamados de entidades e seus **relacionamentos**. O objetivo é apresentar como os objetos estão definidos e como se relacionam uns com os outros. Representa de forma abstrata os dados que vão ser armazenados independentemente da implementação. O Modelo Conceitual se tornou padrão e seus diagramas quando concluídos serão indicativos que a terceira fase pode ser iniciada. O projeto Lógico realiza o mapeamento entre o Modelo Conceitual e o Projeto Físico. O modelo é dependente do tipo de SGBD, podendo ser um SGBD relacional, orientado a objetos, distribuídos, etc., sem depender da distribuição de SGBD. O projeto físico torna o projeto lógico um projeto pronto para a implementação, detalhando campos, índices, tipos de preenchimentos, nomenclaturas etc. Alguns DBAs (*Database Administrator* – ou Administrador de Banco de Dados) que utilizam o SGBD Relacional finalizam a etapa do projeto físico criando

um script com a estrutura interna de armazenamento, utilizando a *Data Definition Language* (DDL) do SQL. A figura 1 apresenta as fases para a construção de um Banco de Dados.

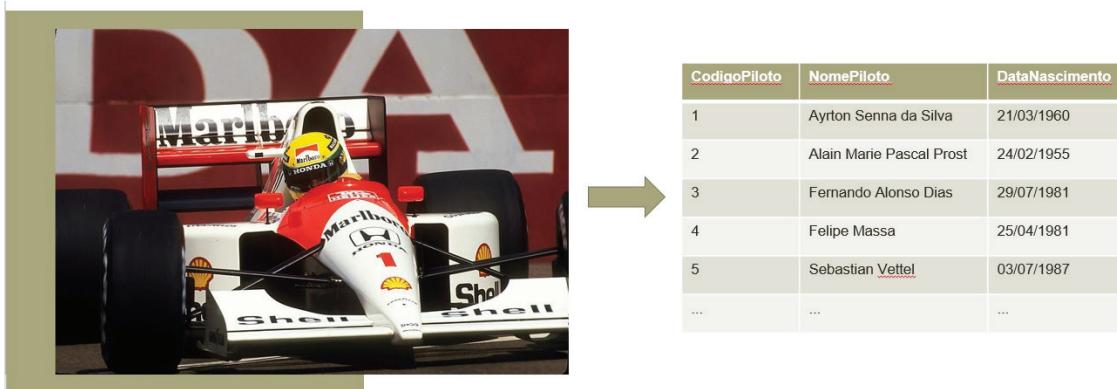


**Figura 1:** Fases para a construção de um Banco de Dados.

É importante o DBA projetar e não pular as fases apresentadas, pois estudos indicam que ao tentar otimizar uma etapa, o profissional gera uma visão equivocada do minimundo e problemas de implementação vão surgir aumentando drasticamente as manutenções no SGBD. Ao seguir as etapas para elaborar um projeto de banco de dados, gera-se uma documentação que irá ajudar a modelagem orientada a objetos. O alto índice de reutilização de código bastaria para convencer a equipe a realizar o trabalho, mas temos uma diminuição significativa na manutenção do SGBD e melhora no desempenho global do sistema.

## ■ MODELAGEM DO E-R

A abordagem relacional representa os dados de forma simples, como um conjunto de tabelas bidimensionais originadas em linhas e colunas. Os dados ficam distribuídos em tabelas e as operações são realizadas por uma linguagem que manipula o conjunto de uma só vez. O princípio é de transparência para o usuário, ou seja, ele não deve e nem precisa saber onde os dados estão armazenados e como estão armazenados. Os usuários manipulam os dados por meio de objetos representados por tabelas. A figura 2 apresenta o objeto piloto de Formula 1 (F1).



### MER

Modelo Entidade-Relacionamento, representa os dados de forma simples, como um conjunto de tabelas bidimensionais.

**Figura 2:** Objeto Piloto de F1 e sua representação em uma tabela

Fonte: [https://commons.wikimedia.org/wiki/File:Ayrton\\_Senna\\_McLaren\\_MP4-6\\_1991\\_United\\_States.jpg](https://commons.wikimedia.org/wiki/File:Ayrton_Senna_McLaren_MP4-6_1991_United_States.jpg)

O modelo de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados representado por um modelo entidade-relacionamento (**MER**) que gera um diagrama entidade-

-relacionamento (**DER**). Utilizar o MER apresenta diversas vantagens como: apresentar de maneira robusta uma sintaxe do modelo de informação de maneira simples e clara, possibilitar a compreensão do modelo pelos usuários com um pouco de esforço, pode-se criar o modelo com facilidade, pode-se inter-relacionar diversos projetos e por não vincular o modelo a um SGBD tem independência de SGBD.

Nosso objetivo é modelar uma representação abstrata do banco de dados contendo **entidades** e relacionamentos que representem as informações do negócio evitando redundâncias, inconsistências e que economizem espaço. É importante conhecer os objetos do negócio que podem ser classificados em dois grupos: entidades e relacionamentos.

As entidades são objetos que podem existir no mundo real e apresentam um significado próprio ou podem ser objetos abstratos para capturar informações que serão armazenadas no banco de dados. Em um DER, as entidades são representadas graficamente por um retângulo com o nome da entidade. Ao nos referirmos à entidade Piloto, falamos de todo o conjunto de pilotos armazenados com seus **atributos**. Se você precisar se referir a um piloto, então utilize a terminologia instância da entidade ou ocorrência de entidade. A figura 3 apresenta a representação gráfica de duas entidades.



**Figura 3:** Representação gráfica das entidades Piloto e Equipe.



pense nisso

Defina quais seriam as entidades que você deveria criar para armazenar os dados das corridas de Fórmula 1. As entidades abstratas são as mais difíceis de definir. Assim, identifique quais seriam essas entidades para a Fórmula 1.

Observe que a entidade Piloto não informa quais pilotos estão armazenados no banco de dados, mas que o banco de dados contém informações dos pilotos. Neste momento, representamos graficamente que se deseja manter dois objetos, mas não definimos quais serão as informações armazenadas. Essas informações serão definidas por relacionamentos, atributos, generalizações ou especializações.

## DER

Diagramas Entidade-Relacionamento.

## Entidade

são objetos que podem existir no mundo real e apresentam um significado próprio ou podem ser objetos abstratos para capturar informações que serão armazenadas no banco de dados.

## Atributos

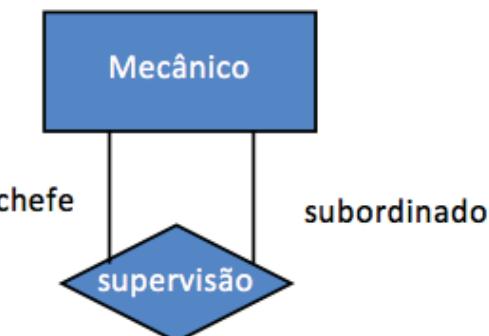
informações que qualificam uma entidade, descrevem suas características, representam propriedades elementares de uma entidade ou relacionamento.

O relacionamento é uma associação entre objetos, representado por um losango ligado por linhas aos retângulos. É caracterizado por um verbo, por exemplo: Pilotos trabalham para uma Equipe. A figura 4 apresenta o relacionamento entre as duas entidades.



**Figura 4:** Representação gráfica das entidades Piloto e Equipe e o relacionamento.

O modelo do banco de dados expressa um conjunto de objetos classificados como pilotos, um outro conjunto de objetos equipes e o conjunto de associações entre pilotos e equipes. Para exemplificar, imagine uma instância específica: o piloto Ayrton Senna da Silva trabalhou na Ferrari. Existem alguns casos em que uma entidade não se relaciona com outra entidade, mas com ela mesma, criando um **auto-relacionamento**. Neste caso, cria-se o conceito de papel da entidade no relacionamento. A figura 5 apresenta um caso de auto-relacionamento.



**Figura 5:** Auto-relacionamento dos Mecânicos.



dica

Todo o relacionamento tem um verbo, sendo representado por um losango. Ao iniciar a modelagem de um banco de dados, relacionar as entidades com um verbo pode se tornar um desafio. Defina como as entidades descobertas anteriormente se relacionam no banco de dados da Fórmula 1, anote, desenhe o diagrama e compare com o feito na aula. Não serão iguais, mas devem lembrar ou remeter a mesma atividade.

# ■ ATRIBUTOS E CHAVES

Os atributos são informações que qualificam uma entidade e descrevem suas características ou elementos. No diagrama físico são chamados de colunas ou campos. É uma característica que se repete. Podem ser classificados como: simples, composto, multivvalorado e especiais. O atributo simples é aquele que não é divisível como o nome do piloto. Os atributos compostos são aqueles que podem ser divididos em sub atributos como por exemplo o endereço que pode ser dividido em logradouro, número, bairro, cidade, estado e cep. Os atributos multivvalorados podem ser organizados como uma lista, conjunto ou coleção.

A maioria das ocorrências é do tipo simples. Os atributos multivvalorados são os casos em que podem aparecer um conjunto de valores, como por exemplo a maioria dos pilotos tem apenas uma cidadania, mas alguns podem ter dupla cidadania. Por fim os atributos especiais são os utilizados para indicar uma função especial como **chave primária**, chave estrangeira, **chave candidata** e chave composta.



## palavra de autor

“Os atributos representam propriedades elementares de uma entidade ou relacionamento.

Cada atributo está associado a um domínio particular, que é um conjunto de valores válidos

para o atributo. [...] São as características de uma entidade que a descrevem detalhadamente.

Uma ocorrência específica de um atributo em uma entidade ou relacionamento denomina-se valor do atributo.” MACHADO (2014 p. 72).

O atributo especial chave primária identifica unicamente a instância entre todas as outras instâncias armazenadas e deve ter conteúdo reduzido e um valor constante no tempo. O atributo especial chave candidata indica que pode vir a ser utilizado como chave primária. O atributo especial chave estrangeira indica que a entidade se relaciona com outra chave primária de outra entidade. O atributo especial chave

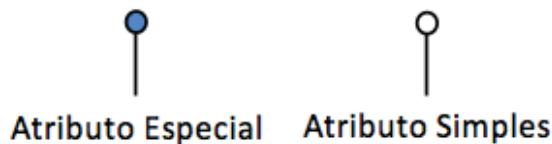
### Chave Primária

identifica unicamente a instância entre todas as outras instâncias armazenadas e deve ter conteúdo reduzido e um valor constante no tempo.

### Chave Candidata

indica que o atributo pode vir a ser utilizado como chave primária.

composta é formado pelo conjunto de diferentes atributos. A figura 6 apresenta os tipos gráficos diferentes que podem ser utilizados para os atributos.



**Figura 6:** Tipos gráficos para atributos.

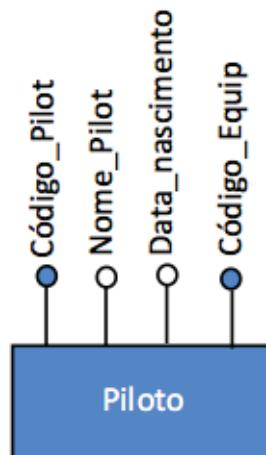
Para ilustrar, a figura 7 apresenta a entidade Piloto e seus atributos. Note que a entidade Piloto se encontra até o momento com uma chave primária Código\_Piloto, uma chave estrangeira Código\_Equipe e dois atributos simples Nome\_Piloto e Data\_nascimento. O processo de modelagem é progressivo e a cada momento o diagrama será revisto e modificado. Não pense que, ao iniciar a modelagem, todos os dados serão descobertos e que você não terá de refazer o diagrama.

### Tuplas

instância de uma entidade ou registro.

**você sabia?**

Alguns autores chamam a instância de uma entidade de registro; contudo, a maioria gosta de se referir a elas como **tuplas**. Você não irá modelar como entidade um objeto que não apresenta diversos registros ou tuplas.



**Figura 7:** Entidade Piloto com seus atributos.



pense nisso

Definir os atributos das entidades expande o diagrama e acrescenta detalhes tornando a visão do banco de dados melhor. A dificuldade é definir os atributos simples, compostos, multivalorados e especiais. Sabendo que é um trabalho árduo, refine o diagrama da Fórmula 1 para que todos os atributos sejam descobertos e grafados corretamente.

## CARDINALIDADE

Os relacionamentos apresentam **cardinalidade** com grau dos relacionamentos que podem ser 1:1 um para um, 1:N um para muitos e N:N muitos para muitos. Em um relacionamento 1:1, um registro de uma entidade se relaciona com somente um registro na outra entidade. A figura 8 apresenta um relacionamento um-para-um.

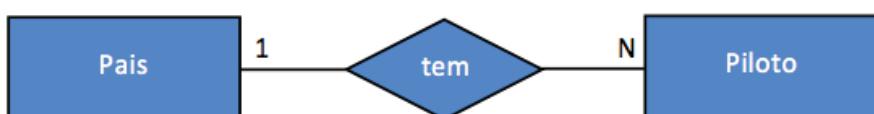
### Cardinalidade

grau dos relacionamentos mínimo e máximo de uma entidade.



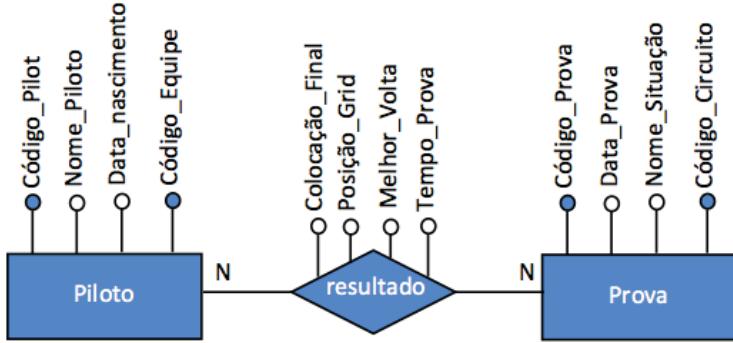
**Figura 8:** Relacionamento 1:1.

No relacionamento 1:N, cada registro de uma entidade se relaciona com muitos registros na outra entidade. É a cardinalidade mais comum encontrada no mundo real e por isso a mais utilizada nos diagramas. A figura 9 apresenta um relacionamento um-para-muitos.



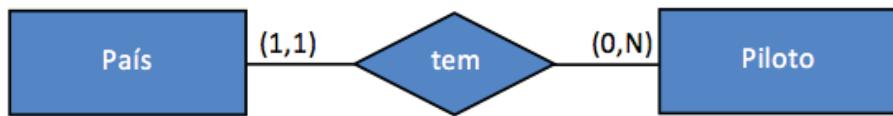
**Figura 9:** Relacionamento 1:N.

O relacionamento N:N representa que vários registros de uma entidade se relacionam com vários registros na outra entidade. Observe que os dados são, neste tipo de relacionamento, atribuídos ao relacionamento e não à entidade. A figura 10 apresenta um relacionamento N:N da Fórmula 1.



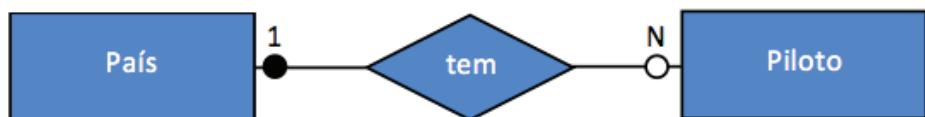
**Figura 10:** Relacionamento N:N.

A cardinalidade pode ser representada com a cardinalidade mínima e máxima. Neste caso, os valores mínimos e máximos são representados entre parênteses seguindo a ordem de primeiro ser indicado o valor mínimo seguido do valor máximo. A figura 11 apresenta o relacionamento entre País e Piloto e indica que um país pode não ter um piloto na F1 ou ter muitos pilotos. O piloto terá obrigatoriamente um país.



**Figura 11:** Relacionamento 1:N com cardinalidade mínima e máxima.

Outra forma de representar cardinalidade mínima igual a zero é apresentada na figura 12. Observe que a cardinalidade mínima 0 é representada por um pequeno círculo aberto indicando que a participação da entidade pode existir ou não, ou seja, sua participação é opcional. O círculo pintado indica que a participação é obrigatória, ou seja, não existirá um piloto que não tenha seu país de origem cadastrado. Mas poderá existir um país sem um piloto.

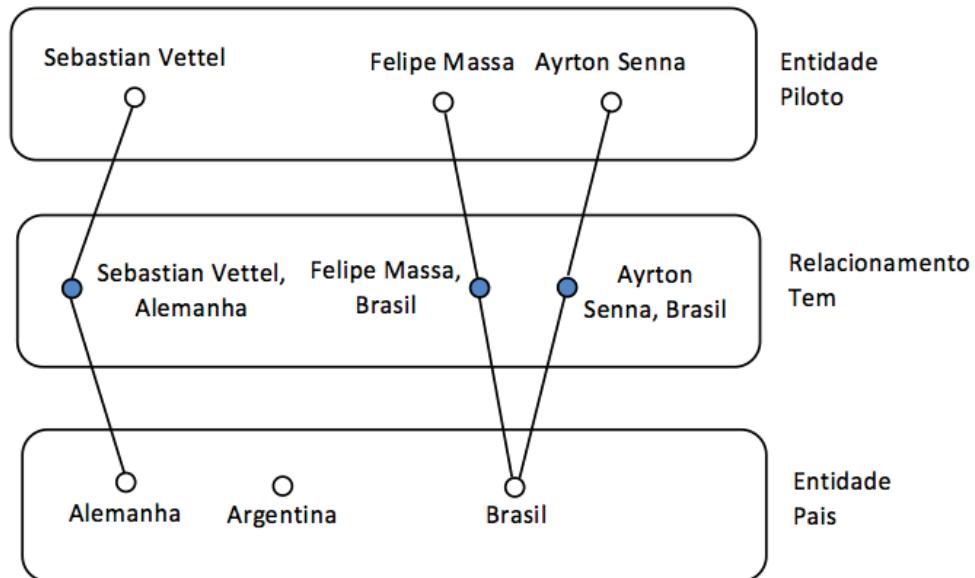


**Figura 12:** Relacionamento 1:N com cardinalidade mínima e máxima.

Agora que você já definiu os atributos das entidades no seu diagrama da Formula 1. Agora revise seu diagrama para grafar a cardinalidade máxima e mínima para todas as entidades.

Para facilitar o entendimento da cardinalidade, você pode criar um diagrama de ocorrência. As ocorrências são representadas grafica-

mente por círculos brancos e as ocorrências de relacionamentos são representadas por círculos azuis. As ocorrências estão interligadas por linhas, sendo apenas permitido um círculo se ligar a um círculo azul. A figura 13 apresenta um diagrama de ocorrência entre Piloto e País.



**Figura 13:** Diagrama de Ocorrência.

Todos os relacionamentos vistos até agora se referem à relacionamentos binários, ou seja, relacionamento entre duas entidades. Embora a maioria dos relacionamentos ocorra entre duas entidades, existem casos em que pode ser necessário um relacionamento com maior quantia de entidades. No caso de participar três entidades em um relacionamento, esse relacionamento é nomeado de ternário e deve ser utilizado com cuidado pois induz à criação de banco de dados não normalizados. A figura 14 apresenta um **relacionamento ternário**.

#### Relacionamento ternário

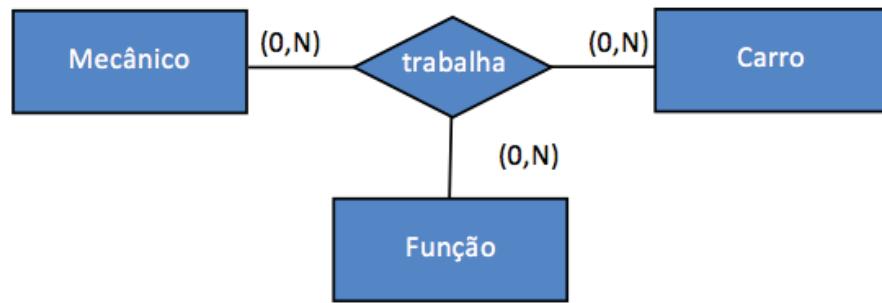
participam três entidades em um relacionamento.

! **atenção**

Como regra, você só deve criar um relacionamento ternário quando não existe forma binária de representar a relação. Na teoria, existe a possibilidade de inúmeras entidades se relacionarem, os chamados **relacionamentos e-nário**, mas no mundo real você não encontra esse tipo de relacionamento.

#### Relacionamentos e-nário

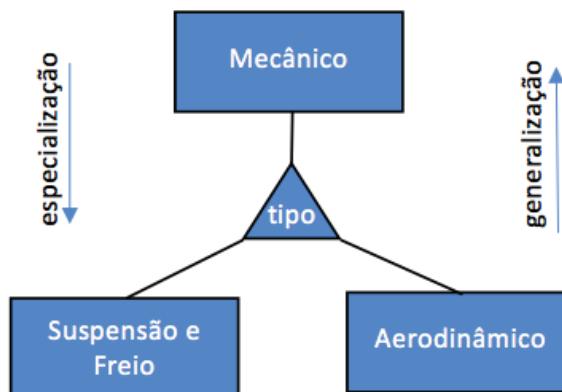
participam inúmeras entidades no relacionamento.



**Figura 14:** Relacionamento ternário.

## ■ ESPECIALIZAÇÃO OU GENERALIZAÇÃO

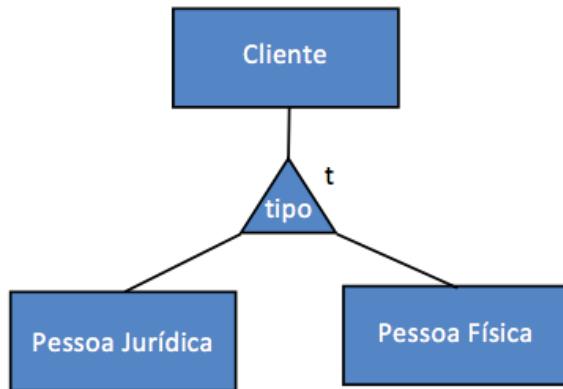
A generalização é uma abstração de entidades semelhantes, possivelmente com vários atributos comuns e apenas alguns diferentes que apresentam a mesma chave primária. São representados graficamente por um retângulo. O conceito de generalização ou especialização se dá de acordo com o sentido que lemos o relacionamento. A figura 15 apresenta uma generalização ou especialização.



**Figura 15:** Generalização ou Especialização.

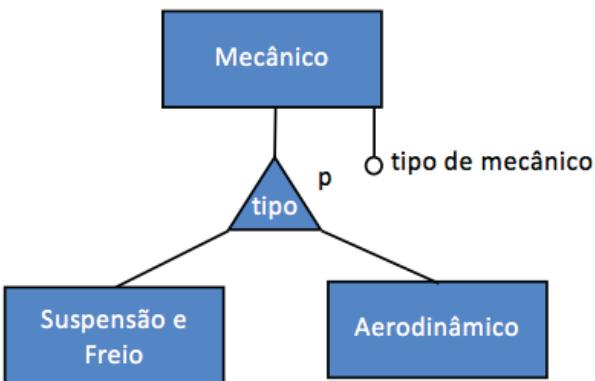
Você deve utilizar generalização ou especialização se duas entidades apresentam vários atributos semelhantes entre si, com pequenas diferenças. Os atributos semelhantes devem pertencer à entidade generalizada e os diferentes pertencem às especializações. A generalização ou especialização pode ser de quatro tipos: total, parcial, exclusiva

ou compartilhada. A generalização é dita como total se for obrigatória a ocorrência. Em uma generalização total sempre existirá uma das ocorrências representadas na entidade especializada. Por exemplo veja na figura 16 a representação do Cliente que será obrigatoriamente pessoa física ou jurídica.



**Figura 16:** Generalização ou Especialização Total.

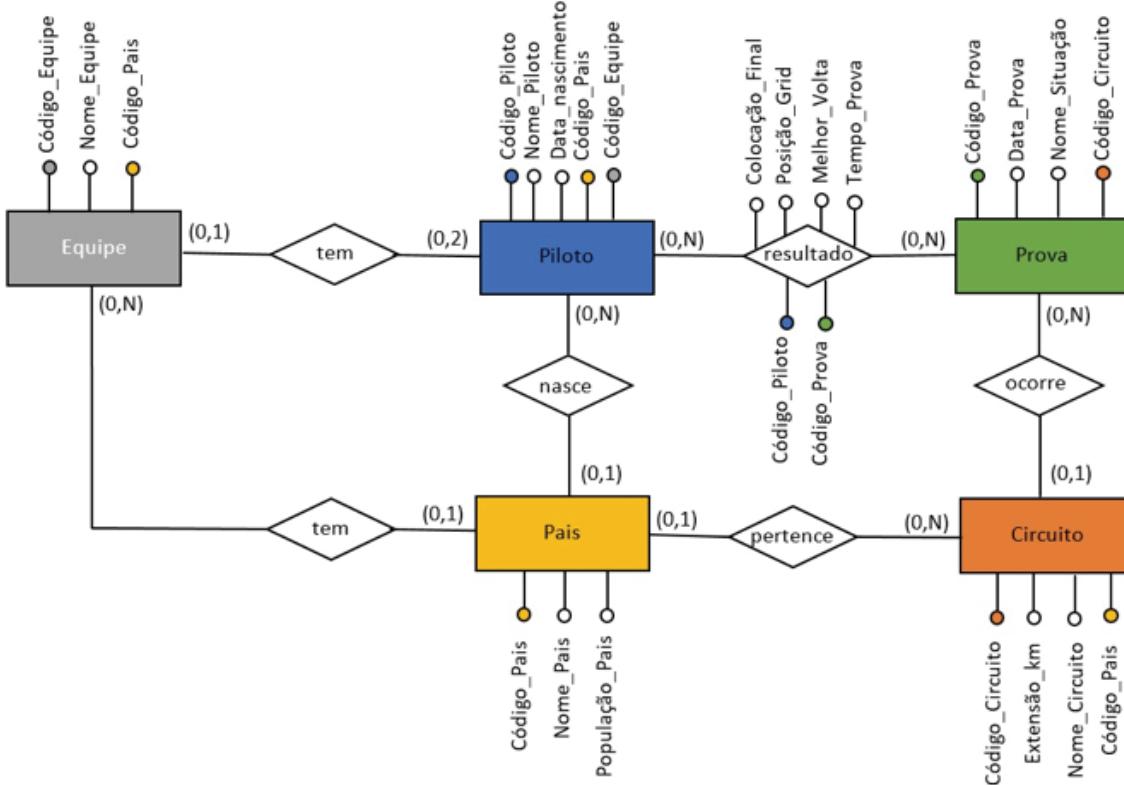
Em uma generalização ou especialização parcial, não é toda a ocorrência da entidade genérica que ocorre a especialização. Na generalização parcial, você deve criar um atributo tipo para indicar a ocorrência na entidade generalizada. A figura 17 refina a generalização da Fórmula 1 como sendo do tipo parcial.



**Figura 17:** Generalização ou Especialização Parcial.

A generalização ou especialização é dita exclusiva quando apenas se permite assumir uma das entidades especificadas, ficando restrita às demais ocorrências. Assim, um mecânico de suspensão e freio nunca poderá assumir o papel de mecânico aerodinâmico, ou seja, ele não pode assumir ambas as funções ao mesmo tempo. Para indicar que a generalização é exclusiva, acrescente junto ao triângulo a le-

tra 'x'. A generalização ou especialização compartilhada permite que uma entidade possa assumir várias especializações e sua indicação gráfica é a letra 'c' junto ao triângulo. Você pode combinar dois tipos em uma generalização sendo permitido total ou parcial com exclusiva ou compartilhada.



**Figura 18:** Diagrama com cores para ilustrar atributos especiais.

Até o momento, você modelou um diagrama parecido com o apresentado na figura 18 que apresenta atributos especiais que nomeamos de chaves primárias e chaves estrangeiras. A figura 18 foi ilustrada colorida para facilitar a diferenciação entre chave primária e chave estrangeira.

Observe que a entidade apresenta a mesma cor que seu atributo especial chave primária.

Quando a chave primária de uma entidade está presente em outra entidade para criar o relacionamento, você está criando um atributo especial chave estrangeira. Sempre que ocorrer um relacionamento 1:N, a entidade N receberá a chave primária da entidade 1, como ocorre no relacionamento entre Equipe e País. Caso o relacionamento seja N:N, o relacionamento receberá as chaves primárias das duas entidades

como ocorre entre as entidades Piloto e Prova. Observe que o relacionamento recebe atributos e se apresentará uma chave primária composta por duas chaves estrangeiras. Todas as entidades modeladas até o momento apresentaram o atributo especial chave primária indicando que identificamos entidades fortes. Contudo, em alguns casos, pode existir entidades que não possuem atributo especial chave primária, indicando que se tornará mais difícil selecionar uma instância ou ocorrência. As **entidades fracas** dependem de outra entidade que se tornará sua entidade proprietária. As entidades fracas sempre vem de relacionamentos 1:N, sendo o 1 a cardinalidade da entidade proprietária e N a cardinalidade da entidade fraca.

Neste exemplo, fica claro a situação da entidade fraca, onde a chave primária do Exemplar é formada pela composição entre a chave da Entidade Carro e o atributo Número do Exemplar. A leitura sempre será dependente da entidade forte. Por exemplo: O carro 2 da Ferrari SF70H têm 728 kg. A entidade fraca apresenta graficamente uma linha de contorno grossa ou uma segunda linha, como indicado na figura 19.

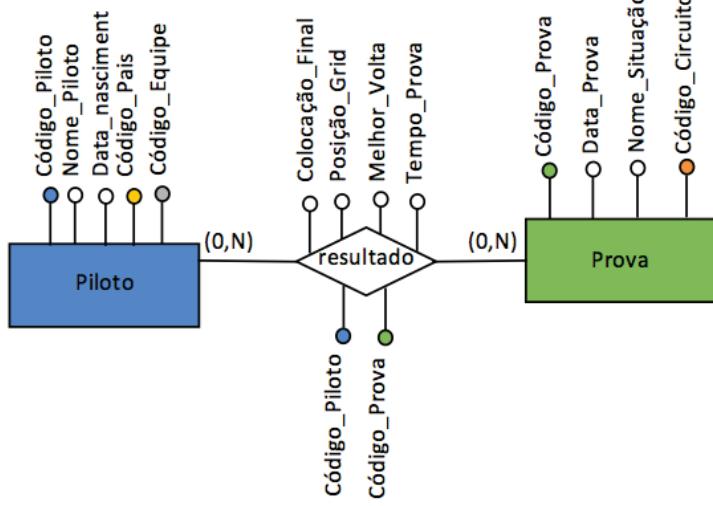
### Entidades Fracas

dependem de outra entidade que se tornará sua entidade proprietária.

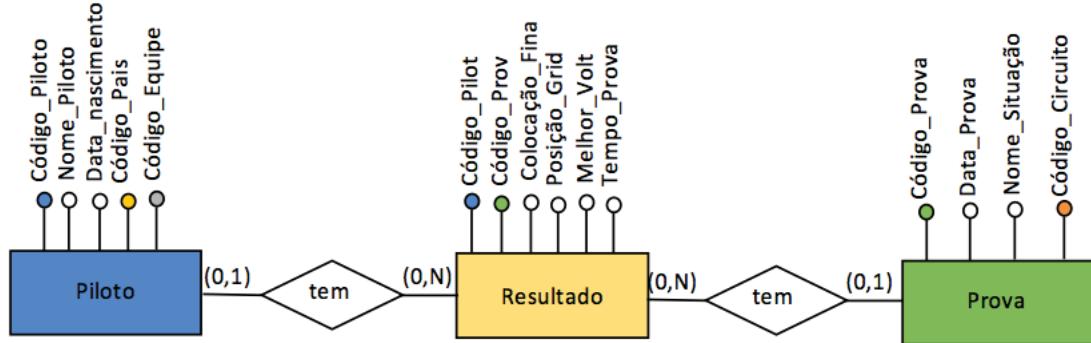


**Figura 19:** Relacionamento 1:N com cardinalidade mínima e máxima.

Ao identificar atributos em um relacionamento, às vezes não fica claro se o relacionamento deveria ser desmembrado em outra entidade ou mantido como relacionamento, e permitir que o relacionamento tenha atributos. Na prática, essas dúvidas serão comuns e diferentes soluções entre os analistas poderão até levar a acirradas discussões sobre como representar a realidade no modelo. Essas discussões são inúteis visto que diferentes modelos podem levar a construção do mesmo banco de dados. Há um conceito de equivalência entre os DER. Podemos dizer que dois DER são equivalentes quando expressam o mesmo problema e modelam a mesma realidade, produzindo assim o mesmo esquema de Banco de Dados. O primeiro caso de equivalência é o representado na figura 20.



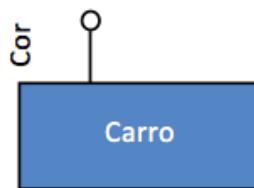
a) Modelagem de relacionamento com atributo.



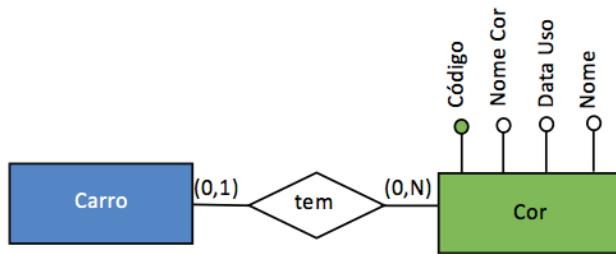
b) Modelagem de nova entidade com relacionamentos.

**Figura 20:** Agrupamento N:N.

Outra discussão é se um determinado objeto deve ser modelado como entidade ou atributo. Para exemplificar, imagine o objeto Cor, este pode ser modelado como atributo da entidade Carro ou como uma entidade Cor relacionada a Carro, veja a ilustração dos dois cenários possíveis ilustrado na figura 21. Ambos os cenários são possíveis, o correto dependerá do que se deseja armazenar. O cenário “a” é correto caso desejemos armazenar apenas uma única cor predominante.



a) Modelagem do objeto Cor como atributo.



- b) Modelagem do objeto Cor como entidade.

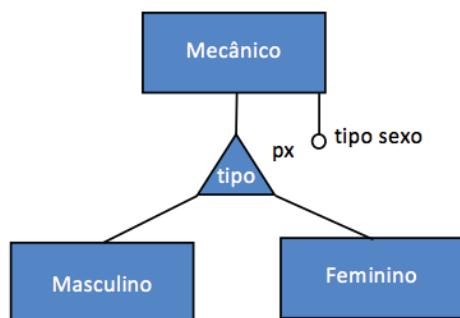
**Figura 21: Possibilidade de Modelagem.**

Caso o objeto que se deseja modelar apresente mais de um atributo, você deve escolher o cenário “b” onde armazenamos diversos atributos. Além disso, se você ao analisar notar que vai existir transações durante a vida da entidade vinculada deve escolher o cenário “b”. No exemplo apresentado na figura 21 se você imaginar que as cores de um carro vão mudar, ou ser inseridas, alteradas ou excluídas, então o melhor cenário é o “b”. Caso você conclua que, enquanto aquele carro existir sua cor será imutável, então o indicado será o cenário “a”.

Outra dúvida na modelagem é se devemos modelar como atributo ou uma generalização. A regra se mantém, se você souber que o objeto modelado vai ter mais de um atributo, nesse caso você deve criar a especialização e não manter como atributo. A figura 22 mostra um exemplo de dois cenários possíveis.



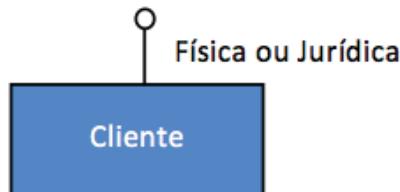
- a) Objeto Sexo modelado como atributo.



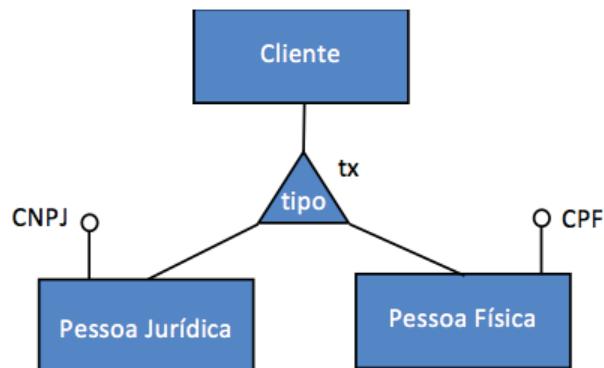
- b) Objeto Sexo modelado como generalização.

**Figura 22: Generalização x Atributo Caso indicado modelar como atributo.**

Observe que o Sexo do Mecânico é melhor representado pelo cenário "a" pois não acrescentamos nenhum atributo nas especializações. Vai existir ocorrências para atributos diferentes dos especificados no cenário "b". Mas não podemos fazer disso uma regra, a escolha pode ser diferente dependendo do caso, como exemplo veja a figura 23 que apresenta um caso em que a escolha indicada é a generalização.



- a) Objeto Física ou Jurídica modelado como atributo.



- b) Objeto Física ou Jurídica como Generalização.

**Figura 23:** Generalização x Atributo Caso indicado modelar como generalização.

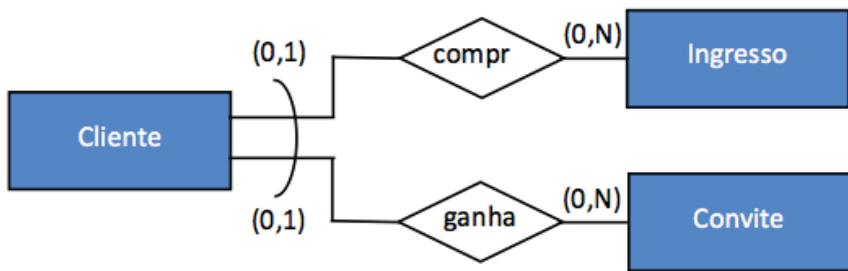
## ■ CONDICIONALIDADE DE RELACIONAMENTO

### Relacionamento condicional

se um relacionamento ocorre com uma entidade este relacionamento será exclusivo e não será mais permitido os demais relacionamentos.

Um relacionamento pode ter restrições ou ser condicional. Na literatura pode ser nomeado de **relacionamento condicional** ou relacionamento exclusivo. Você pode criar um relacionamento com duas ou mais entidades, mas permitir que se um relacionamento ocorrer com uma das entidades, este relacionamento será exclusivo e não será mais permitido os demais relacionamentos. O uso de um relacionamento condicio-

nal ocorre quando você deseja representar algumas regras de negócio. A figura 24 apresenta um modelo de relacionamento exclusivo.

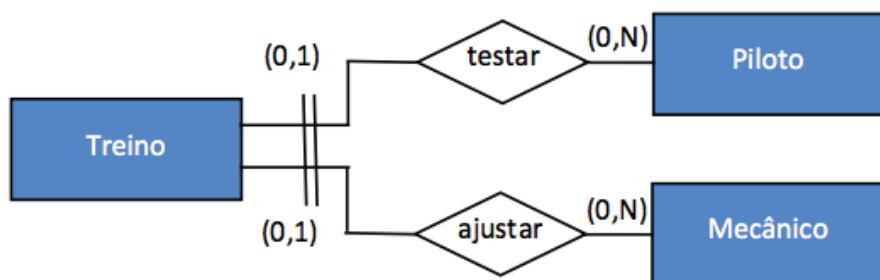


**Figura 24:** Condicionalidade de Relacionamento.

No relacionamento apresentado na figura 24, ou o cliente compra ingressos ou ele ganha convites. Observe que, se uma condição ocorrer, a outra não poderá ocorrer.

## ■ RELACIONAMENTOS CONTINGENTES

Este tipo de relacionamento permite você criar relacionamentos que devem ocorrer ao mesmo tempo com entidades diferentes. Você estabelece que só existe o relacionamento se simultaneamente for atribuído à todas as entidades. A figura 25 apresenta um exemplo de relacionamento de contingência.



**Figura 25:** Condicionalidade de Relacionamento .

No exemplo, é impossível realizar um Treino sem os Pilotos e Mecânicos, sendo obrigatório a ocorrência de ambas as instâncias para que o treino ocorra.

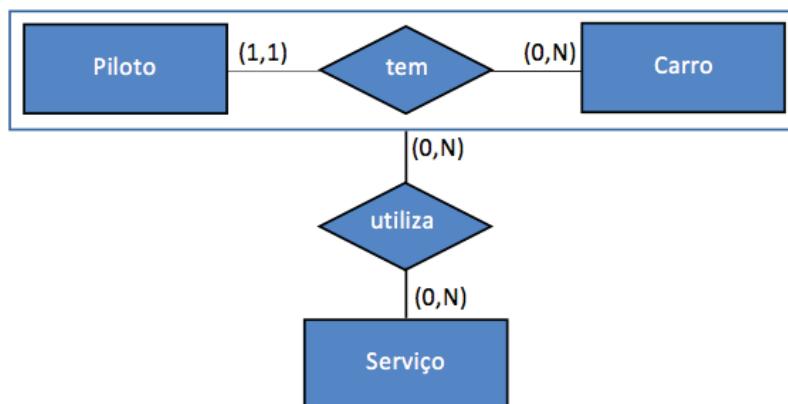
# ■ AGREGAÇÃO

## Agregação

encapsular duas entidades e seu relacionamento dentro de uma agregação e este por sua vez poderá se relacionar com outra entidade.

É um mecanismo de abstração em que você cria a partir de um relacionamento entre duas entidades. Normalmente, você usará a regra de **agregação** toda vez que sentir a necessidade de criar uma relação com outra relação.

A norma não permite você criar uma relação com outra relação, assim, você deve encapsular as duas entidades e seu relacionamento dentro de uma agregação e esta, por sua vez, poderá se relacionar com outra entidade. A figura 26 apresenta o conceito de agregação. Vale lembrar que quando você usa agregação, a entidade nova só vai existir após a existência do fato, ou seja, o relacionamento interno entre as duas entidades.



**Figura 26:** Exemplo de Agregação.

# ■ VALIDAÇÃO DE DIAGRAMAS

## Erros sintáticos

quando não segue as regras de construção do diagrama.

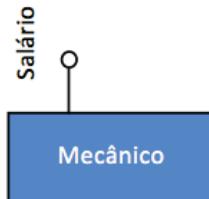
Ao finalizar a modelagem de um diagrama, você deve validar se o mesmo está correto, completo e sem redundâncias. O modelo pode conter **erros sintáticos** e semânticos. Os erros sintáticos ocorrem quando você não segue as regras de construção do diagrama, como por exemplo associar atributos à atributos, associar relacionamentos por meio de outro relacionamento, ou especializar relacionamento ou atributo. Você pode evitar esses erros ao utilizar ferramentas de modelagens que não permitem.

tam essas ações. Os **erros semânticos** são aqueles que ocorrem quando você não cometeu erro de sintaxe mas refletiu a realidade de forma inconsistente. Como exemplo, associar o atributo CNPJ à entidade Cliente em vez de associá-lo à especialização Pessoa Jurídica. Para validar se o modelo está completo você deve validar: se todas as propriedades estão presentes no banco de dados, se é possível extrair todos os resultados esperados do banco de dados e se é possível modificar ou registrar as transações no banco de dados. A última validação é se existem conceitos redundantes, ou seja, não deve existir relacionamentos que sejam combinações de outros relacionamentos com a mesma entidade.

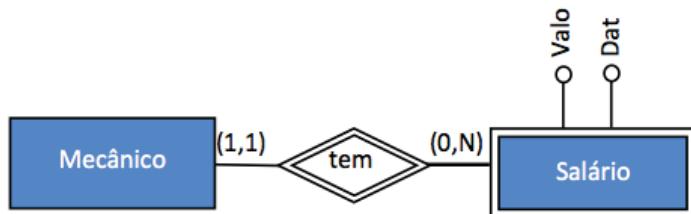
**Erros semânticos**  
quando refletiu a  
realidade de forma  
inconsistente.

## ■ ASPECTO TEMPORAL

Ao modelar um banco de dados é comum modelar um banco funcional sem se preocupar com o que vai ocorrer com o tempo de uso. É importante se perguntar quais informações são úteis no decorrer do tempo e quais podem ser eliminadas. Lembre-se que um banco de dados não poderá crescer infinitamente e que algumas informações poderão ser eliminadas. Um exemplo desta decisão é apresentado na figura 27, você deve avaliar se você deve ou não manter um histórico dos salários dos mecânicos.



- a) Modelagem do objeto Salário como atributo.

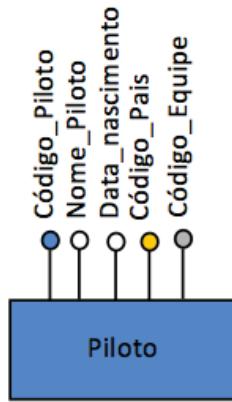


- b) Modelagem do objeto Salário como entidade.

**Figura 27:** Possibilidade de Modelagem leve em conta o aspecto temporal.

# ■ MODELO LÓGICO

Após a fase de Modelagem Conceitual e construção do DER, você inicia a construção do modelo lógico. Ele vai levar em consideração a tecnologia existente no SGBD mas deve ser portável entre os SGBDs existentes ou similares. O modelo lógico representa a estrutura de dados conforme visto pelo usuário. O modelo deve definir quais as tabelas que serão criadas e os nomes das colunas. Como exemplo, observe parte do diagrama DER criado na fase de Modelagem Conceitual.



**Figura 28:** Modelagem Conceitual de entidade Piloto com seus atributos.

O modelo conceitual apresentado na figura 28 é expresso no modelo lógico no formato texto assim:

Piloto(codigo\_piloto, nome\_piloto, data\_nascimento, codigo\_pais, codigo\_equipe)

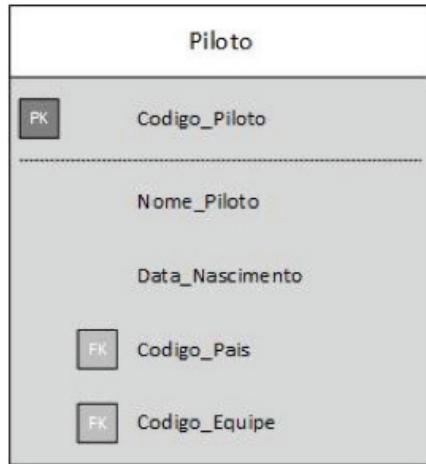
Observe que o nome da entidade aparece primeiro e os atributos estão dentro dos parênteses, sendo a chave primária identificada com o sublinhado. O mesmo elemento no formato gráfico é apresentado na figura 29. A notação de Engenharia da Informação é empregada para representar o DER por muitas **Ferramentas CASE** como notação de modelagem de dados.

Os relacionamentos são representados apenas por uma linha que liga as entidades com símbolos representando a cardinalidade. A notação só permite relacionamentos binários entre entidades, sendo os relacionamentos ternários modelados como relacionamentos binários. Os atributos só podem aparecer em entidades, sendo necessário transformar relacionamentos com atributos em entidades.

## Ferramentas Case

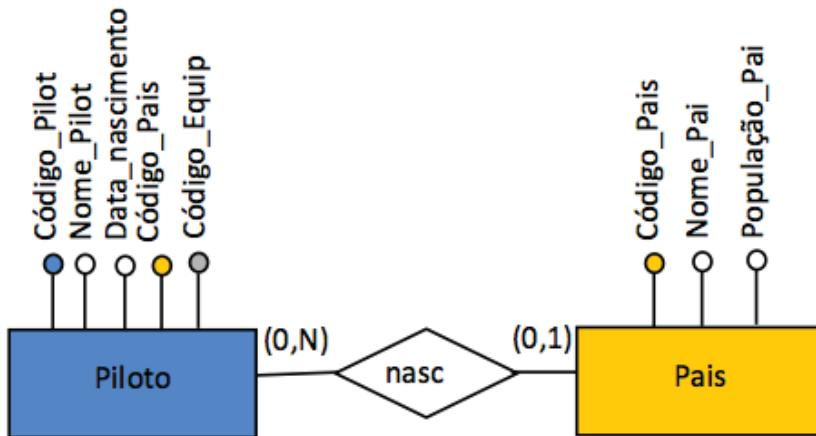
Ferramentas de Modelagens de Banco de Dados automatizadas que visam auxiliar o desenvolvedor.

Os relacionamentos N:N serão modelados como entidades. Os relacionamentos 1:N irão provocar a criação de um atributo na entidade N com a chave primária da entidade 1. Os relacionamentos 1:1 na maioria das vezes vão gerar uma fusão de entidades só ocorrendo exceção quando for (0,1) em ambos os lados do relacionamento que neste caso gera uma adição de coluna em uma das entidades.



**Figura 29:** Modelagem Lógica da Entidade Piloto com seus atributos.

Observe outra parte do diagrama do Modelo Conceitual, agora com duas entidades Piloto e País apresentada na figura 30.



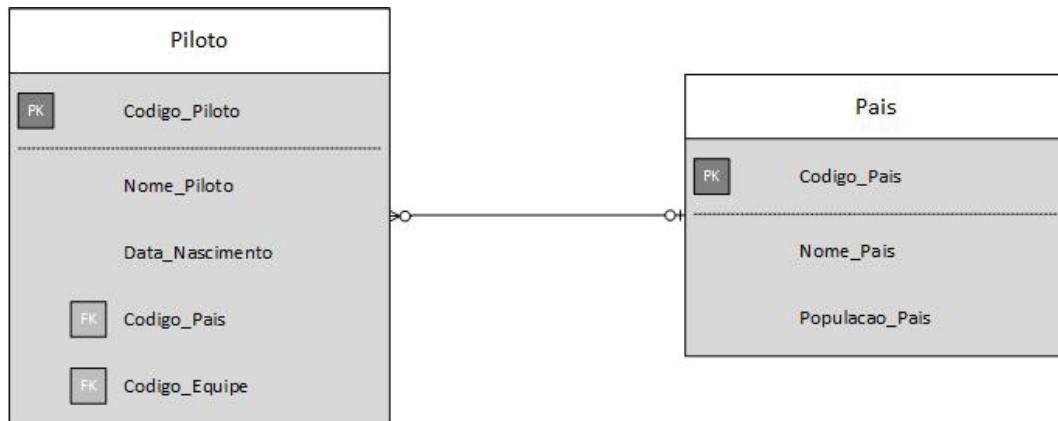
**Figura 30:** Modelagem Conceitual das entidades Piloto e País.

O modelo conceitual apresentado na figura 30 é expresso no modelo lógico no formato texto assim:

Piloto(código\_piloto, nome\_piloto, data\_nascimento, código\_país, código\_equipe)

País(código\_país, nome\_país, poulacao\_país)

O mesmo elemento no formato gráfico é apresentado na figura 31.

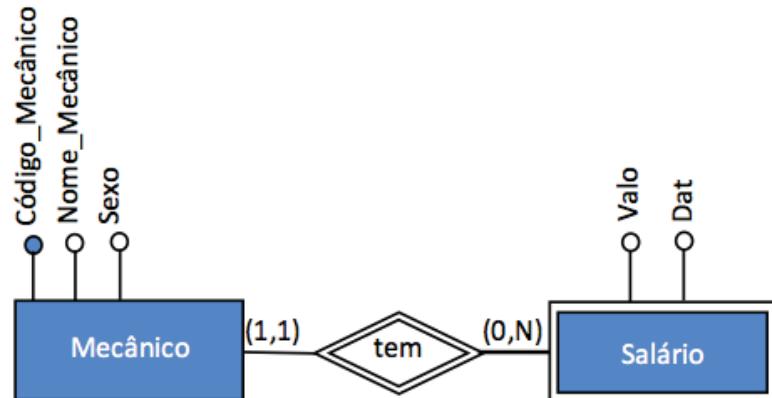


**Figura 31:** Representação das entidades Piloto e País com notação Engenharia da Informação.

A tabela 01 apresenta a notação mínima e máxima para cardinalidade com notação Engenharia da Informação.

Notação	Cardinalidade
+	1
-○-	0
↖	N
†	(1,1)
○-	(0,1)
↖	(1,N)
○↖	(0,N)

Veja o Modelo Conceitual apresentado na figura 32 com uma entidade forte e uma entidade fraca como ficam representados no Modelo Lógico.



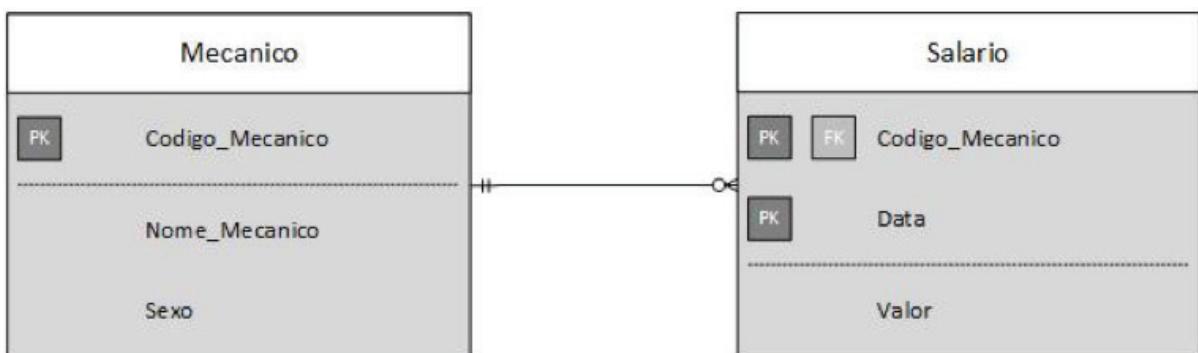
**Figura 32:** Modelagem Conceitual das entidades Mecânico e Salário

O modelo conceitual apresentado na figura 32 é expresso no modelo lógico no formato texto assim:

Mecanico(codigo\_mecanico, nome\_mecanico, sexo)

Salario(codigo\_mecanico, data, valor)

O mesmo elemento no formato gráfico é apresentado na figura 33. Observe que a entidade fraca apresenta a chave primária composta por dois elementos, a chave estrangeira que é chave primária na entidade forte da qual a entidade fraca depende e mais um atributo que permite a identificação única da tupla ou instância.



**Figura 33:** Representação das entidades Mecânico e Salário com notação Engenharia da Informação.

A notação Engenharia da Informação é simples e fácil de utilizar. O título da entidade deve ser o nome da entidade que você utilizou. Na notação Engenharia da Informação, para identificar os atributos chaves primários você utiliza a abreviação de *Primary Key* (PK). Os atributos do tipo chave estrangeira são representados por *Foreign Key* (FK). Lembre-se que uma chave primária é uma identificação única, nunca irá se repetir e que não aceita valores nulos. A chave estrangeira é um atributo que aponta para a chave primária de outra entidade e nesta entidade pode se repetir. No caso, apresentado na figura 33 existe uma chave primária composta. Na entidade Salario o conjunto de atributos que compõem a chave não pode se repetir, ou seja, podemos ter vários salários atribuídos a um mecânico, mas nunca irá ocorrer de no mesmo dia o mecânico ganhar duas mudanças em seu salário. Assim, o mecânico só poderá sofrer uma alteração em determinada data, podendo o mesmo mecânico ter outros registros de alterações de salários em datas diferentes.

# ■ REGRAS DE INTEGRIDADE

## Integridade

manter dados consistentes e refletindo a realidade.

Um dos principais objetivos dos SGBDs é manter a **integridade** de dados. Dizer que o Banco de Dados não está íntegro é o mesmo que dizer que seus dados não refletem a realidade e que seus dados não são consistentes entre si. Para conseguir manter a integridade, os SGBDs oferecem mecanismos para garantir a integridade. Uma restrição de integridade é uma regra de consistência que é assegurada pelo SGBD. Existem quatro tipos de classificação para as regras de integridade: domínio, chave, vazio e referencial.

A integridade de domínio é quando o SGBD garante que exista somente dados no atributo correspondente ao tipo predefinido. Para exemplificar, imagine o atributo `codigo_piloto`, sendo definido como inteiro, só deverá permitir números inteiros e não deverá aceitar caractere ou data. São comuns erros de integridade de domínio ao se tentar importar dados entre SGBDs diferentes, sendo necessário, na maioria das vezes, alguns ajustes para tornar compatível os domínios existentes.

A integridade de chave é quando os atributos chave primária e chave candidata são verificados, garantindo que não exista valores nulos e que todos os valores atribuídos sejam únicos permitindo o acesso de forma inequívoca à tupla.

A integridade referencial é quando existe atributo chave estrangeira em uma entidade. Os atributos de chaves estrangeira devem ser verificados e validados como pré-existentes como chave primária em outra entidade. Além disso, os atributos de chave estrangeira devem ter o mesmo domínio dos atributos da chave primária na entidade referenciada. É importante lembrar que a chave estrangeira pode ter valores nulo ou valores existentes na entidade referenciada.

A integridade de vazio verifica se o atributo pode ser ou não vazio. Os atributos que não são permitidos utilizar valores vazios são chamados de campos obrigatórios. Os atributos nos quais podem aparecer valores vazios são chamados de opcionais. Normalmente os SGBDs relacionais exigem que todos os atributos chave primária sejam obrigatórios, sendo que para os atributos chave estrangeira são permitidos valores vazios.

É importante notar que uma vez definida as entidades e estabelecidas as regras de integridade, o SGBD deve manter os dados íntegros. Isso quer dizer que ele não deve permitir que se quebre as regras estabelecidas, proibindo operações que o tornariam não íntegro. Um exemplo disso seria, após estabelecer as regras e inserir dados, apagar uma tupla que é chave primária de uma entidade que a contém como chave estrangeira. O SGBD não deve permitir que alguém apague a tupla enquanto a mesma for referência em outra entidade. O mesmo deve ser feito na atualização de dados e inserção. As quatro restrições de integridades definidas devem ser garantidas automaticamente pelo SGBD relacional, sem exigir que o programador escreva um procedimento para garantí-las explicitamente. Ainda é possível que o SGBD controle outras restrições de integridade. As restrições de integridade semântica são aquelas que o usuário define. É o SGBD que deve garantir que só entrem dados que foram validados pela regra que o usuário definiu. Um exemplo seria na entidade Mecânico definir que o atributo sexo só permita o cadastro de 'M' ou 'F' no atributo. Sendo M para Masculino e F para Feminino. Além de verificações, é possível criar regras semânticas para funções, procedimentos, gatilhos e definir predicados próprios.



## síntese

Nesta aula, você conheceu as etapas de um projeto de Banco de Dados. Você tomou conhecimento e pode compreender como funciona o Modelo Entidade-Relacionamento (MER) para bancos de dados. Você viu como criar o Diagrama Entidade-Relacionamento (DER). Você aprendeu a criar DER incorporando os conceitos de integridade, chave primária, chave estrangeira e cardinalidade. Você entendeu o que é integridade e está apto a treinar muito para fixar o conhecimento adquirido nesta aula. Lembre-se que modelar e programar é igual a andar de bicicleta, por mais que você veja alguém fazendo, só sentirá a dificuldade e fixará conhecimento quando praticar.

## Atividades

- 01.** Analise e crie o Modelo Conceitual para um Banco de forma que permita cadastrar nome do banco, código do banco, os dados dos correntistas, como nome do correntista, data de nascimento, sexo, endereço completo do correntista, os dados da conta, tipo do correntista, dígito verificador da conta, tipo da conta, valor do saldo, número da agência, nome da agência, endereço completo da agência. No endereço, coloque o nome do logradouro, tipo, título, nome do município, estado e cep.

- 02.** Tendo como base o Modelo Conceitual do Banco, converta para o Modelo Lógico Escrito.

Tipo\_Conta(id\_tipo\_conta, nm\_tipo\_conta)

Banco(id\_banco, nm\_banco)

Conta(id\_conta, id\_agencia, id\_tipo\_conta, id\_municipio, nr\_digito, vl\_saldo)

Agencia(id\_agencia, id\_banco)

Municipio(id\_municipio, id\_estado, nm\_municipio)

Logradouro(id\_cep, id\_municipio, nm\_logradouro, ds\_titulo)

Estado(id\_estado, nm\_estado)

Correntista(id\_correntista, id\_cep, nm\_correntista, dt\_nascimento, ds\_sexo, nr\_endereco)

Pertence(id\_correntista, id\_conta)

Crie os diagramas de Engenharia da Informação do Modelo Lógico Escrito para o Banco.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

03. Considere o banco de dados com o seguinte esquema

Aluno(rgm\_aluno, nm\_aluno, nm\_pai, nm\_mae, dt\_nascimento, ds\_sexo)

Matricula(id\_classe, rgm\_aluno, dt\_matricula)

Classe(id\_classe, ano\_base, serie, turma, id\_escola, id\_grau, id\_periodo)

Escola(id\_escola, nm\_escola, endereco, bairro)

Grau(id\_grau, nm\_grau)

Periodo(id\_periodo, nm\_periodo)

Usando a notação apresentada nesta aula, construa o esquema diagramático para o banco de dados com a notação de Engenharia da Informação.

**04.** Com base na atividade 4, construa o DER.

## Anotações

# AULA 4

## AMBIENTE DE TRABALHO

### NESTA AULA

- » Introdução à Máquinas Virtuais
- » Instalar um servidor Linux
- » Instalar o SGBD Oracle
- » Para instalar SQL Developer.

### ■ METAS DE COMPREENSÃO

- » Conhecer Máquinas Virtuais.
- » Aprender como criar um servidor Linux e instalar um SGBD Oracle com uma ferramenta para administrá-lo.

### ■ APRESENTAÇÃO

O objetivo nesta aula é apresentar uma ferramenta que lhe auxiliará ao criar um Banco de Dados Relacional. Você vai conhecer algumas ferramentas para administrar e verá como instalar um SGBD relacional. Aprender a instalar um SGBD lhe dará uma noção do tempo e do esforço necessários para criar um servidor de banco de dados. Você notará que existe alguns desafios em preparar um Banco de Dados para funcionar.

Para entender estes conceitos, esta unidade está organizada da seguinte forma:

- » A seção 2 apresenta as Máquinas Virtuais;
- » A seção 3 ensina a instalar um servidor Linux;
- » A seção 4 apresenta como instalar o SGBD Oracle;
- » A seção 5 demonstra como instalar o SQL Developer da Oracle;

Não deixe de utilizar as bibliografias associadas à unidade. Bom estudo!

# ■ INTRODUÇÃO À MÁQUINAS VIRTUAIS

## VM

Máquina Virtual ou Virtual Machine é um aplicativo que executa programas de computadores como um computador real.

A máquina virtual (*Virtual Machine - VM*) é um aplicativo multi-plataforma para computadores que executa programas de computadores como um computador real. Uma VM pode ser definida como um *software* que cria uma cópia de um sistema físico, sendo esta cópia totalmente protegida.

O fato da VM ser multi-plataforma significa que ela pode ser instalada em Windows, Linux, Solaris ou MacOS. A VM deve simular outra máquina com a capacidade de executar outro sistema operacional em um computador. A memória RAM do computador, processador e outros recursos serão todos virtualizados. A virtualização é a capacidade do aplicativo utilizando camadas em isolar o *hardware* e compartilhá-lo com múltiplos ambientes de execução. Após a instalação do aplicativo de virtualização, você poderá criar máquinas virtuais, com disco rígido virtual e executar sistemas operacionais inteiros. Assim você poderá testar sistemas operacionais, ferramentas de desenvolvimento e SGBDs diferentes. Poderá validar a arquitetura que pretende montar para a empresa, montar protótipos e projetos futuros. Em alguns casos poderá inclusive reduzir custos com *hardware*. Como desvantagem, você precisará de um computador com poder de processamento médiano, podendo apresentar demanda de memória RAM.

## Host

a máquina que terá o aplicativo que cria VM instalado

Uma definição importante será o nome que seu computador, ou seja, a máquina que terá o aplicativo que cria VM instalado, recebe – máquina hospedeira ou **host OS**. A máquina criada no aplicativo, ou seja, a máquina virtual recebe o nome – máquina visitante ou **guest OS**.

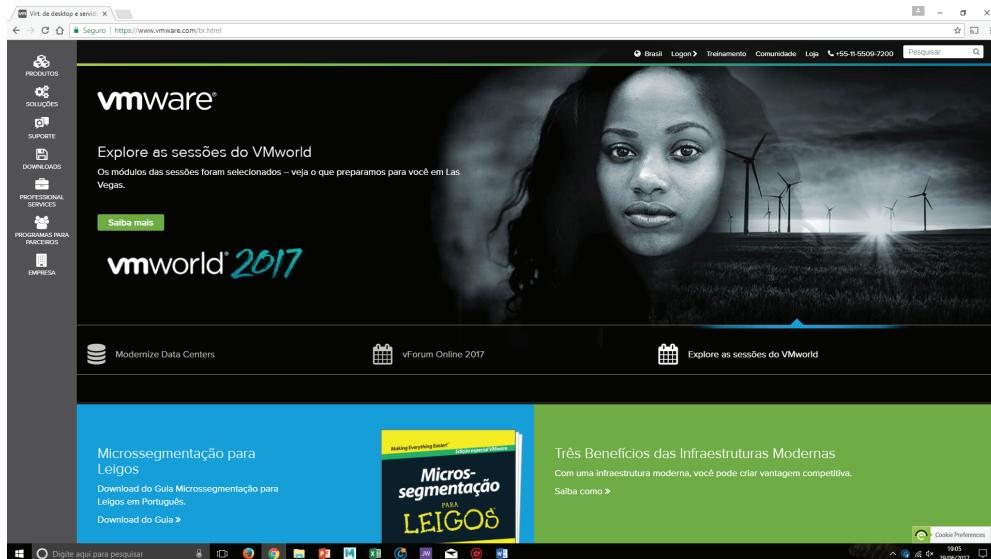
## GUEST

máquina criada no aplicativo VM

A VM, para iniciar a aplicação, precisa de um sistema operacional que dará o suporte para o emulador. Contudo, após iniciar o serviço, a máquina virtual irá rodar de maneira independente do *host*, isso quer dizer que se a *guest* pegar um vírus não irá infectar o *host*. Os arquivos salvos no *guest* não serão visualizados no *host*. Assim, é seguro rodar e realizar teste em máquinas virtuais, pois essas podem ser restauradas facilmente. Existem diferentes aplicativos que permitem criar VM mas

vamos destacar dois o VMWare Player da VMWare e o Virtualbox da Oracle. Ambos são gratuitos e podem ser utilizados sem fins comerciais. O Virtualbox apresenta uma interface mais completa permitindo mais flexibilidade para criar suas máquinas enquanto o VMWare Player apresenta melhor *performance* em execução com um pouco menos de consumo da memória RAM do host. Ambas soluções precisaram que o host tenha habilitado em sua BIOS o suporte a *Virtualization Technology* – **VT** ou Tecnologia de Virtualização. Cada fabricante de hardware cria uma forma de acessar a BIOS e, em alguns casos, variam inclusive de acordo com o modelo do computador. Assim, você deve fazer uma pesquisa na internet em diferentes fóruns de manutenção com os dados de seu PC e verificar a forma de acessar a BIOS do host e habilitar (*Enable*) o suporte da VT. A alteração deve ser salva e seu computador irá iniciar sem nenhuma modificação aparente. Você só notará a diferença ao instalar o aplicativo de virtualização.

Para facilitar vamos focar na aplicação VMWare Player. Acesse a página [www.vmware.com/br.html](http://www.vmware.com/br.html) para realizar o download do arquivo. A figura 01 apresenta a identidade do site.

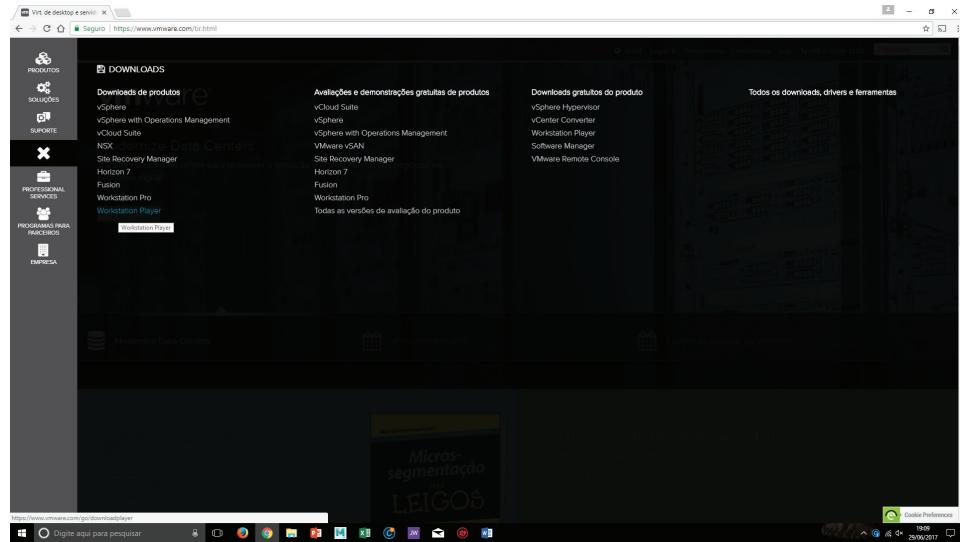


**Figura 01:** Identidade do site [vmware.com.br](http://www.vmware.com.br)

Selecione no menu lateral *download* o submenu *Workstation Player* conforme indicado na figura 02.

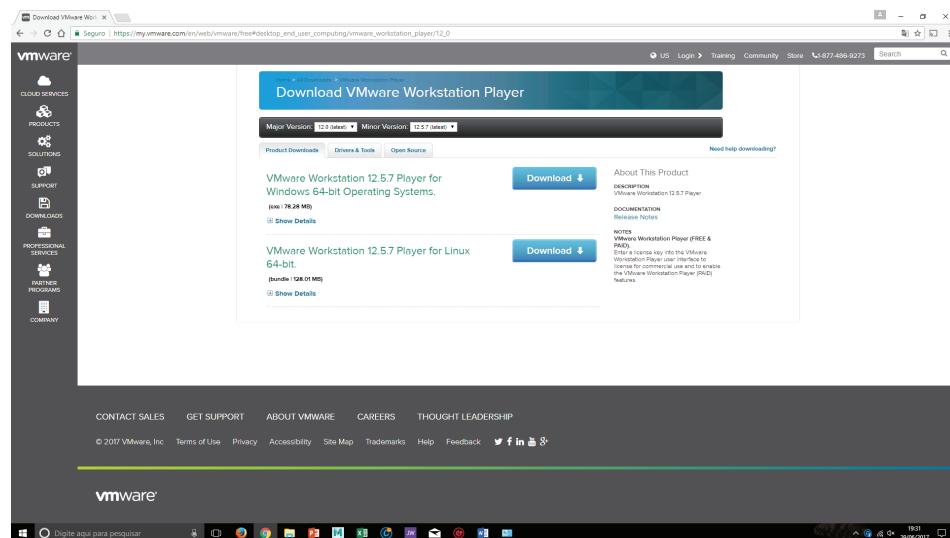
## VT

Virtualization Technology ou Tecnologia de Virtualização que deve ser habilitado na BIOS



**Figura 02:** Seleção da página download Workstation Player do site [vmware.com.br](http://vmware.com.br).

Você será direcionado para uma página como ilustrada na figura 03. Selecione a versão indicada para o seu sistema operacional. O sistema operacional deve ser 64 bits para suportar a instalação da VM. Caso você não saiba se o seu sistema é de 32 bits ou 64 bits, você pode selecionar o botão Iniciar do Windows e na caixa de pesquisa insira Meu Computador e com o botão direito do mouse selecione Propriedades, observe o tipo de sistema se aparecer x64 é 64 bits.



**Figura 03:** Escolha da Versão – Windows ou Linux.



dica

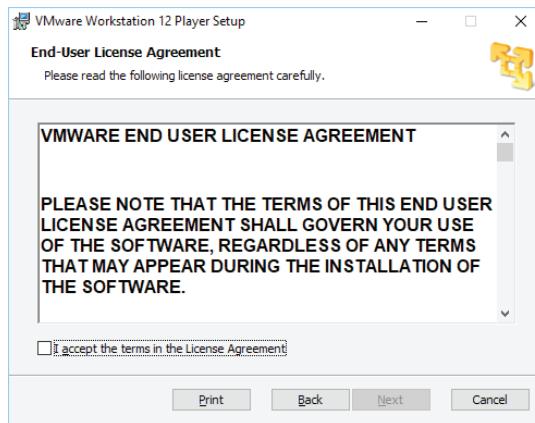
Se o seu sistema for 32 bits, você pode baixar a Virtualbox do site <https://www.virtualbox.org/> no menu Downloads escolha Virtualbox Binaries a versão compatível com o seu sistema operacional.

Após baixar o arquivo, inicie o processo de instalação executando o arquivo. A figura 04 apresenta a tela após a execução do arquivo.



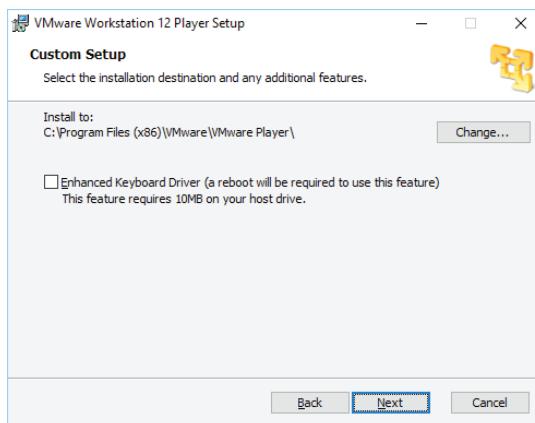
**Figura 04:** Início da instalação da VMware Workstation Player.

Selecione o botão *Next* para continuar o processo de instalação. A seguir você será questionado se aceita os termos e licença de uso da aplicação. A figura 05 apresenta a tela do aceite dos termos e condições de uso. Vale lembrar que a licença para a *VMware Workstation Player* é gratuita para uso não comercial, assim você pode aprender e utilizar em seu computador sem problemas, ficando restrito apenas o uso em empresas e projetos comerciais. Selecione o aceite "*I accept the terms in the License Agreement*" ou "*Aceito os termos do Contrato de Licença*". Após a seleção do aceite, avance pressionando o botão *Next* ou próximo.



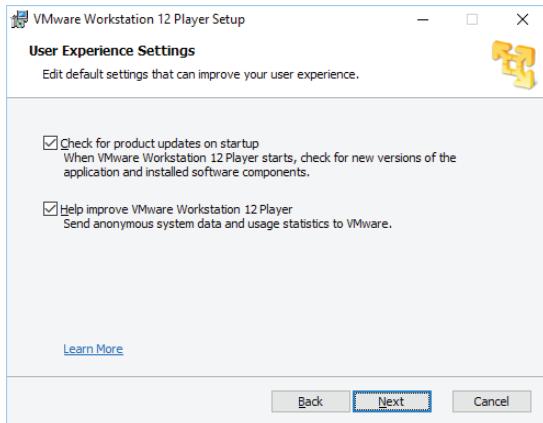
**Figura 05:** Termo de aceite do contrato de licença.

A tela apresentada na figura 06 será exibida solicitando a definição do diretório de instalação. Na maioria dos computadores não é necessário alterar o caminho indicado. Se você precisar alterar selecione o botão *Change* e informe o caminho para a instalação.



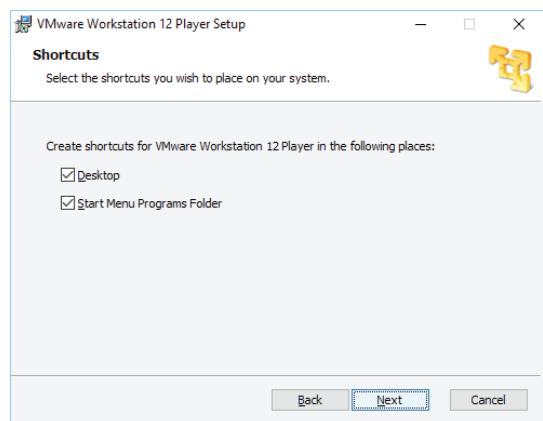
**Figura 06:** Caminho padrão da Instalação da VMWare.

Após aceitar a sugestão do caminho de instalação, você verá a tela apresentada na figura 07 perguntando se você deseja que a VMWare procure por atualizações quando iniciar e se você autoriza que o software envie dados para estatística do fabricante de forma anônima. Após sua escolha seleciona o botão *Next*.



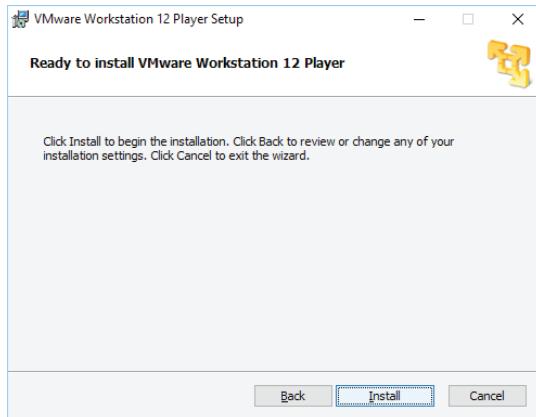
**Figura 07:** Solicitação de Configuração da VMWare.

A seguir a figura 08 irá aparecer perguntando se você deseja que se crie um atalho no *Desktop* e um atalho no menu Iniciar do Windows. Selecione as opções e avance pressionando o botão Next.



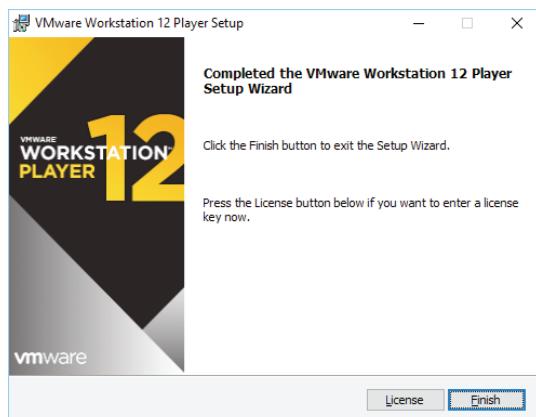
**Figura 08:** Opção de criar atalhos ao instalar.

Após selecionar e definir todas as opções, a tela apresentada na figura 09 irá lhe apresentar a possibilidade de voltar e alterar alguma escolha, cancelar a instalação ou instalar. Você deve selecionar *Install* para iniciar a instalação.



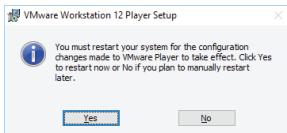
**Figura 09:** Confirmação das opções e início do processo de instalação.

Ao concluir a instalação, você é direcionado para a tela da figura 10 onde você pode finalizar o processo de instalação selecionando o botão *Finish* ou se optar pelo uso comercial que é pago selecionar o botão *License* para digitar a licença de uso comercial. Você deve neste momento selecionar o botão *Finish* uma vez que estamos fazendo uso não comercial.



**Figura 10:** Finalização do processo de instalação da VWware Workstation.

Após finalizar, o software irá solicitar que se reinicie seu computador. A figura 11 apresenta a tela perguntando se deve reiniciar imediatamente ou se você prefere fazer posteriormente. Se selecionar Yes, seu computador irá reiniciar, mas se você estiver ocupado com outras atividades e preferir, pode escolher No e reiniciar após concluir suas atividades. A VM só deverá ser executada após concluir a reinicialização.



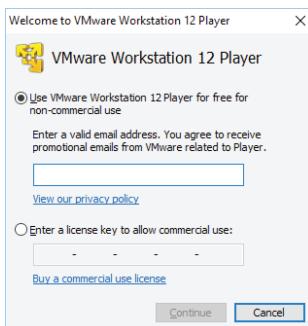
**Figura 11:** Seu computador precisa reiniciar.  
Reinicio agora se selecionar Yes.

O ícone apresentado na figura 12 foi criado na área de trabalho, para acessar o aplicativo da VM você deve dar um duplo clique nele.



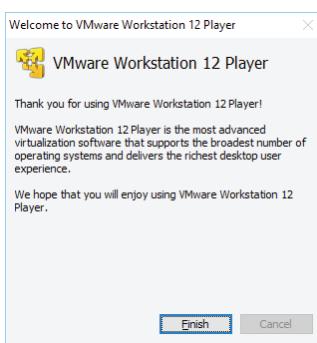
**Figura 12:** Ícone da Ferramenta.

Quando rodar pela primeira vez a aplicação de virtualização, o software irá perguntar sobre o tipo de licença que você está usando, a grátis ou a comercial. A figura 13 apresenta a tela, note que você pode optar pelo uso grátis, não comercial. Digite seu e-mail para prosseguir.



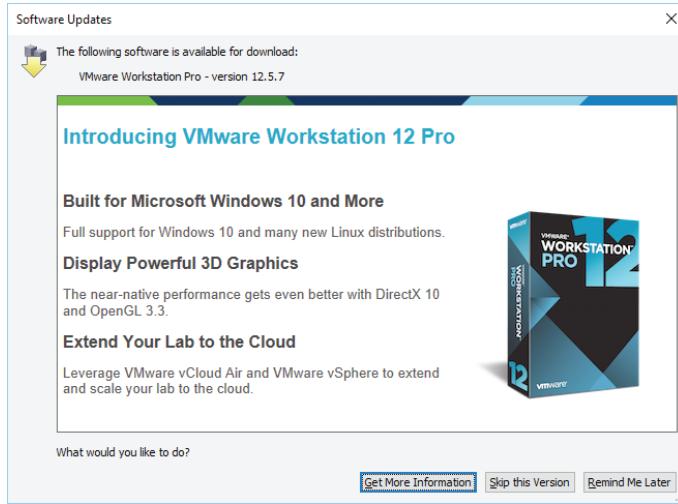
**Figura 13:** Solicitação do cadastro para usar a ferramenta como não comercial.

A tela apresentada na figura 14 irá lhe dar Boas Vindas à ferramenta. Selecione o botão *Finish* para usufruir da ferramenta.



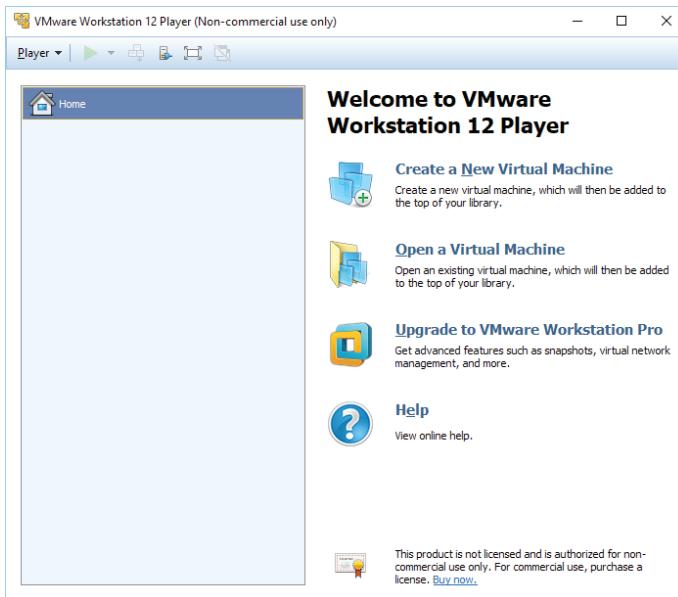
**Figura 14:** Boas vindas da ferramenta.

A seguir, você será convidado a fazer o download da versão *Workstation 12 Pro* ou seja, uma versão mais completa de uso comercial. Seleciona o botão *Skip this Version* para pular o convite e ser direcionado para a ferramenta que você instalou.



**Figura 15:** Convite para utilizar a versão *Workstation 12 Pro*.

Por fim, a interface da ferramenta é exibida na figura 16. Observe que apresenta uma área *Home* onde exibirá a lista de máquinas virtuais existente, que neste momento se encontra vazia. Ao lado, apresenta 4 opções, sendo que nos interessamos pelas duas primeiras, *Create a New Virtual Machine* ou Criar Nova Máquina Virtual e *Open a Virtual Machine* ou Abrir a Máquina Virtual.



**Figura 16:** Interface da *VMware Workstation*.

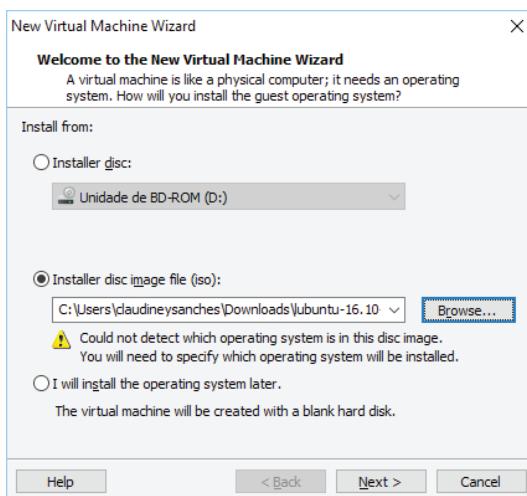
# ■ INSTALAR UM SERVIDOR LINUX

Neste momento você tem a VMWare instalada e pronta para usar. Nela, você poderá criar e instalar diferentes sistemas operacionais e criar diferentes máquinas. O objetivo agora será criar um servidor Linux com o SGBD Oracle em um ambiente gráfico com a ferramenta SQL Developer. Para iniciar, selecione o botão *Create a New Virtual Machine* e a tela apresentada na figura 17 irá abrir solicitando a informação de onde está o disco de inicialização. Nesta aula será apresentado a *distro Lubuntu* por ser leve e rodar com interface gráfica em computadores com pouca memória e pouco poder de processamento. Você pode baixar a imagem em <http://lubuntu.me/>. Selecione a versão que é compatível com sua máquina 32 bits ou 64 bits.



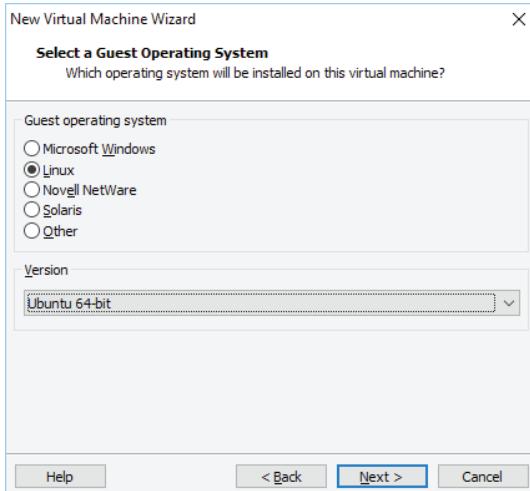
você sabia?

Você pode utilizar diferentes distribuições do Linux para criar seu servidor. O nome das distribuições utilizado na internet é *distro* e facilmente você encontra diferentes *distro* do Linux apresentando filosofias diferentes, interfaces e uma coleções de ferramentas diferentes. Cada grupo que mantem uma *distro* cria ferramentas visando um público.



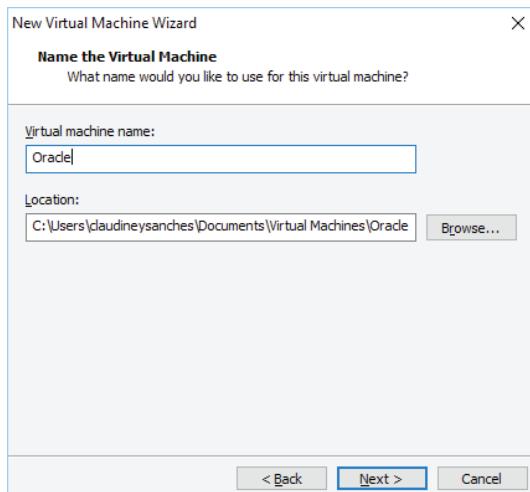
**Figura 17:** Informação do disco de inicialização ou arquivo .iso.

Após informar onde está o arquivo .iso em sua máquina, selecione o botão *Next* e a ferramenta irá solicitar informações do sistema que será instalado na *guest*. A figura 18 apresenta a escolha para o *Lubuntu* que será o *Ubuntu 64-bit* uma vez que são similares.



**Figura 18:** Seleção do SO que será instalado na *guest*.

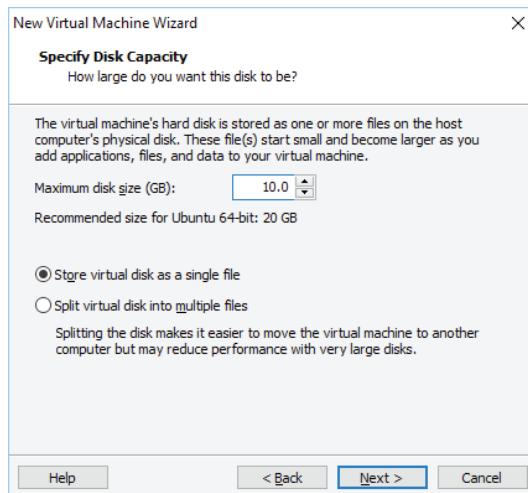
A figura 19 mostra a tela solicitando o nome e o caminho onde serão criados os arquivos da VM. Após o término da instalação, você poderá compactar essa pasta para manter um *backup* da VM. Assim, se alguma coisa der errado com a VM, você poderá restaurar por apagar a pasta e descompactar todos os arquivos.



**Figura 19:** Nome da VM e o caminho onde ela ficou salvo.

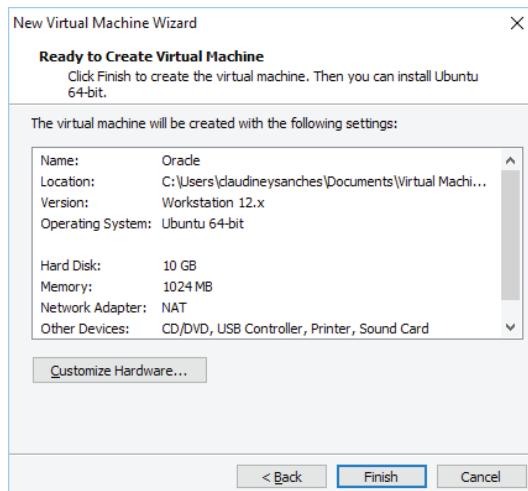
Ao selecionar o botão *Next*, você verá a tela apresentada na figura 20 que solicita o tamanho do disco virtual. O tamanho é dinâmico o que significa que conforme a VM vai precisando de espaço de arma-

zenamento, o sistema vai aumentando o tamanho do arquivo do HD. Assim, recomendo manter a sugestão de 20 GB. Outra pergunta é se o disco será armazenado em um único arquivo ou em múltiplos arquivos. Se o SO do host for uma versão antiga do Windows, como XP, pode ser recomendado utilizar múltiplos arquivos visto que esse sistema operacional utiliza FAT e não NTFS. Se for Windows 7, 8 ou 10, é possível que esteja utilizando sistema NTFS de arquivo o que permite um único arquivo.



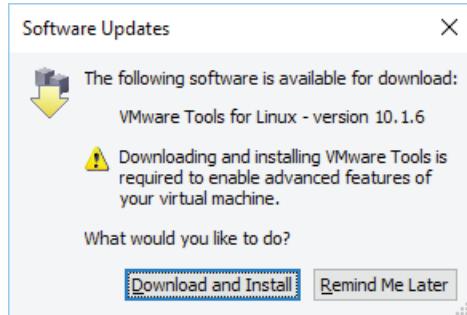
**Figura 20:** Tamanho do HD e definição de um único arquivo para o disco.

Depois de definir todos os parâmetros, a confirmação das especificações da máquina virtual que você está criando será solicitada. A figura 21 apresenta a tela de confirmação. Selecione *Finish* para finalizar o processo e criar a nova VM.



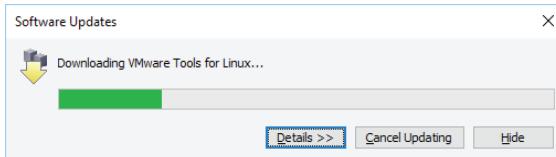
**Figura 21:** Tela de confirmação das opções para criar a nova VM.

Pode ser que o software precise de atualização e por isso, ao finalizar, você seja informado sobre as atualizações nos pacotes Linux para instalar no *guest*, como exemplificado na figura 22, selecione *Download and Install* para baixar as atualizações.



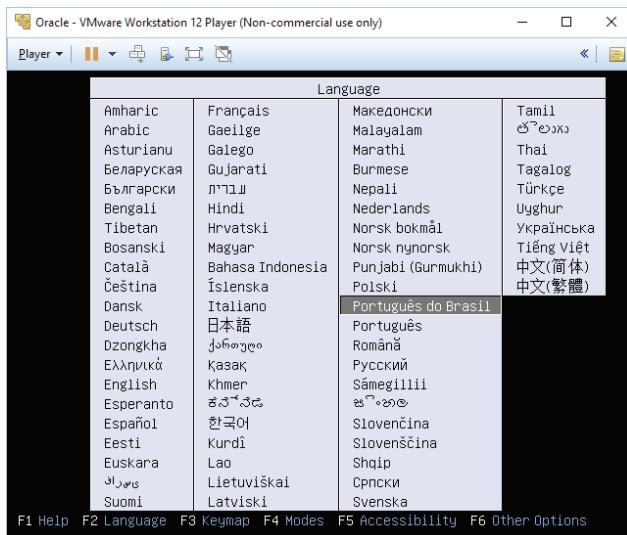
**Figura 22:** Atualizações dos pacotes da VM.

Os pacotes serão baixados para serem instalados no *guest*. Esse processo poderá demorar um pouco, dependendo da velocidade de sua internet. A figura 23 apresenta a tela do *Download* dos pacotes da ferramenta.



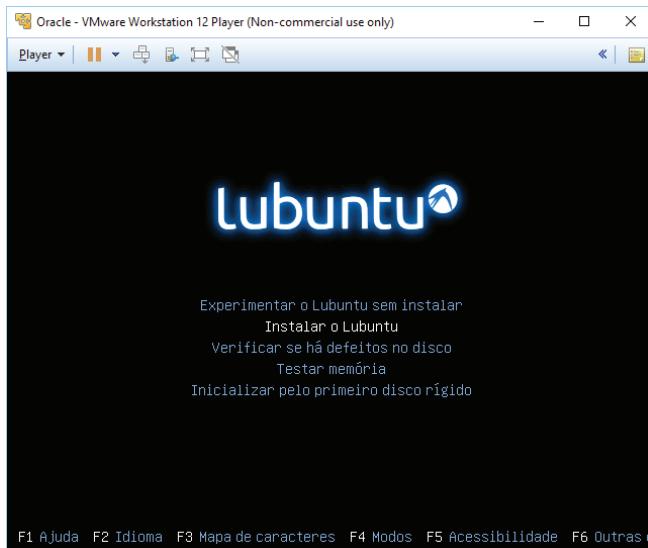
**Figura 23:** Download dos pacotes da VMware.

A VM vai iniciar com o processo de *boot* em uma nova janela. Como indicamos o disco de *boot* da imagem ISO do Linux, logo você verá a figura 24 solicitando o idioma que será utilizado pelo Linux. Seleciona *Português do Brasil*, você poderá utilizar o teclado ou mouse, se quiser sair da janela da VM pressione as teclas *<Ctrl><Alt>* simultaneamente para liberar o mouse do *guest* e voltar ao *host* e *<Ctrl><Alt><Enter>* simultaneamente para alternar entre janela ou tela maximizada no *guest*.



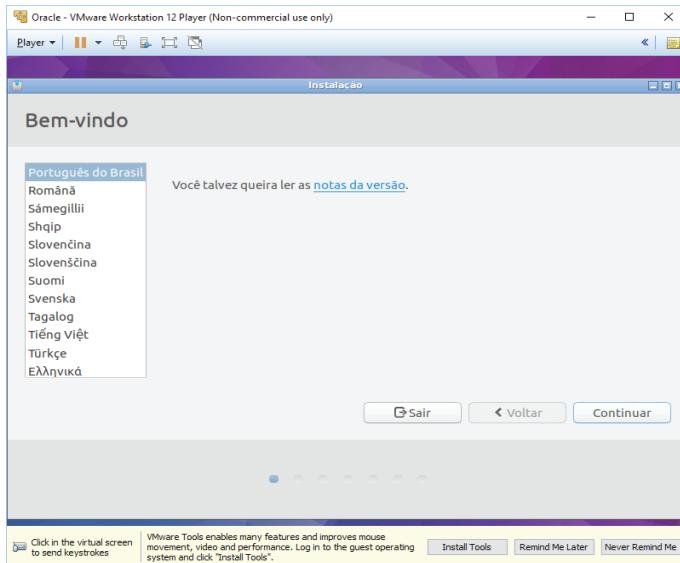
**Figura 24:** Idioma que o Linux irá utilizar.

Uma vez definido o idioma, o Linux irá exibir algumas opções de boot apresentadas na figura 25. Você pode testar o *Lubuntu* antes de instalar, verificando a compatibilidade. Após o teste, reinicie a VM, selecione o idioma e selecione a opção Instalar o *Lubuntu* para prosseguir com a instalação.



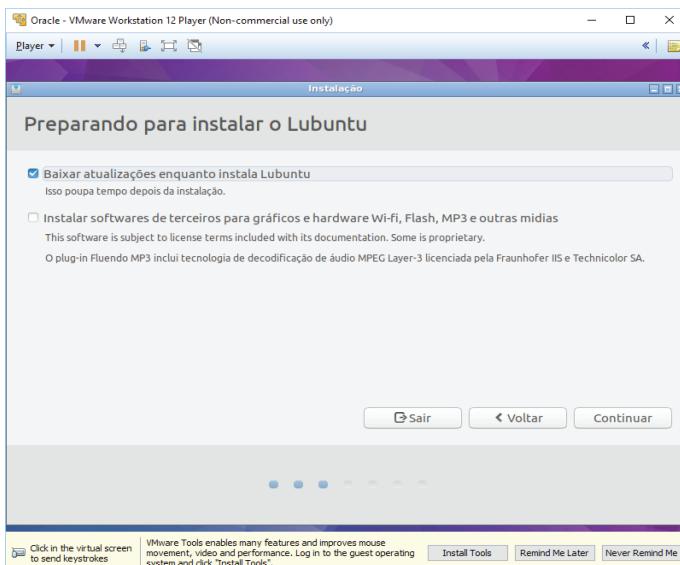
**Figura 25:** Opção de Boot do Lubuntu.

O software de instalação irá carregar um módulo gráfico básico e irá solicitar o idioma de instalação do Linux como apresentado na figura 26. Selecione O Português do Brasil E Continuar para avançar na instalação.



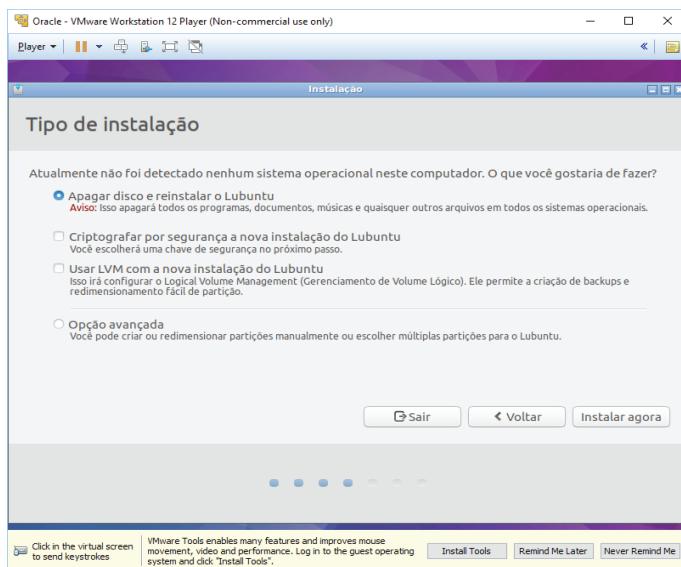
**Figura 26:** Seleção do idioma que será instalado o Linux.

O software de instalação irá solicitar se você quer que ele verifique e atualize o SO durante a instalação, conforme apresentado na figura 27. Esta opção leva a uma instalação mais demorada, mas que facilita o processo de instalação por fazer de uma vez a verificação das atualizações. Se você não tiver problemas de conexão com a internet, recomendo selecionar esta opção, que foi a opção utilizada nesta aula. Alguns podem preferir deixar a atualização para outro momento, neste caso caberá a você conduzir esse processo mais tarde. Independente de sua escolha, selecione **Continuar** para avançar a instalação.



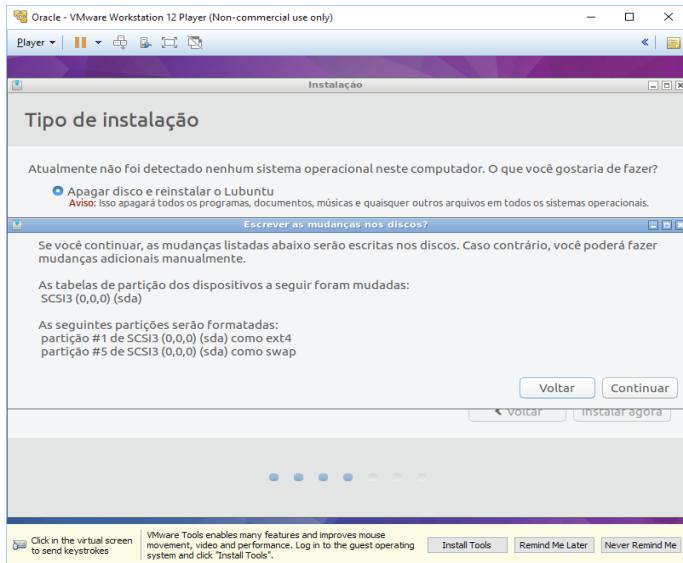
**Figura 27:** Solicitação de autorização para atualizar os arquivos de instalação.

O próximo passo na instalação é definir as partições do Linux no disco, veja figura 28. Os usuários avançados podem querer definir como será criado os discos e definir as partições como *swap*, sistema, *home* e *backup*. Como muitos estão experimentando instalar pela primeira vez o Linux, você vai optar por deixar o software de instalação criar os discos de forma transparente, ou seja, ele deve fazer o processo automaticamente para você, para isso selecione *Apagar disco e reinstalar o Lubuntu*. Esta opção não irá apagar seus dados ou qualquer arquivo no *host*, pois a VM criou um arquivo que simula um disco para essa máquina e ele que vai ser formatado para instalar o Linux.



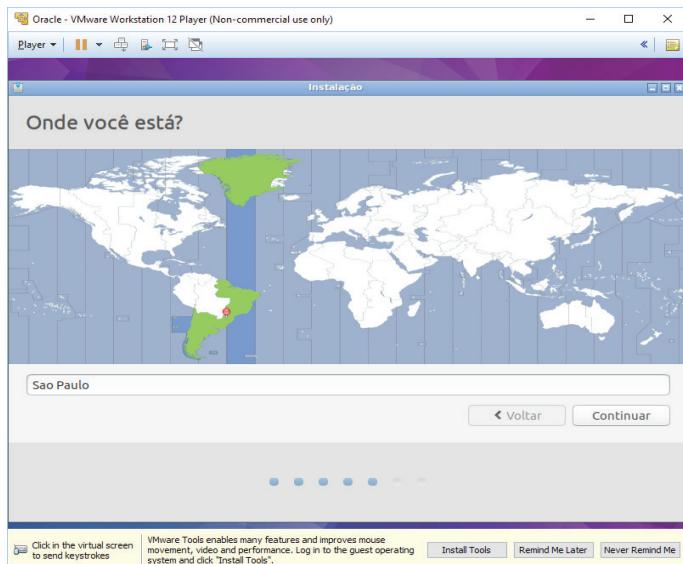
**Figura 28:** Definição dos discos e partições.

Uma janela irá se sobrepor a anterior solicitando que você confirme a ação. A figura 29 apresenta a janela solicitando que se confirme a modificação da partição antiga (*sda*) e para criar a partição *ext4* que conterá o sistema e *home* e uma partição *swap* para acelerar e auxiliar a memória RAM da máquina. Na teoria, a partição de *swap* deve ter o dobro do tamanho de sua RAM, por exemplo se sua VM tem 1GB de RAM a *swap* deve ter 2GB, pois quando sua máquina hibernar, a máquina passa toda a memória RAM para a *swap*. Selecione *Continuar* para avançar para o próximo passo.



**Figura 29:** Confirmar as ações de alteração da partição.

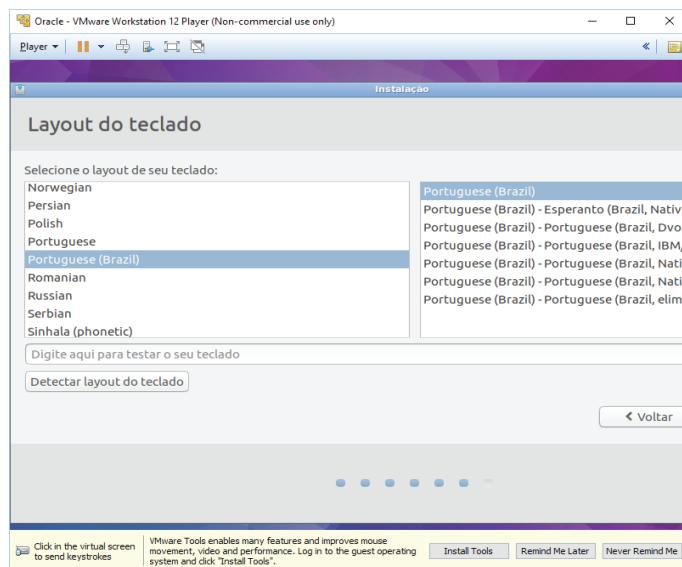
A próxima tela é apresentada na figura 30. O instalador do Linux vai solicitar sua localização. Você pode digitar o nome da cidade e observar no mapa a indicação de sua localização. No caso desta aula, a cidade indicada foi São Paulo. Definir corretamente sua localização facilita em algumas configurações de teclado, fuso horário, moeda e dados do servidor. Após informar a cidade, selecione o botão **Continuar** para avançar a instalação para o próximo passo.



**Figura 30:** Informando o local do Sistema Operacional.

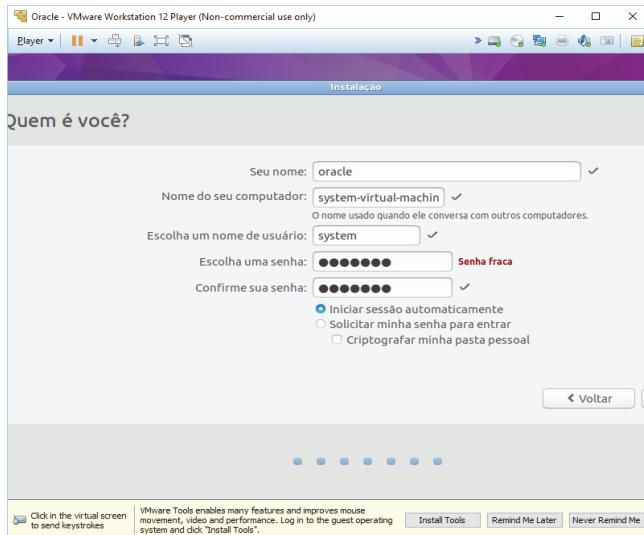
A seguir, você será solicitado a informar o *Layout* do teclado, conforme demonstra a figura 31. Observe que não é a definição de idioma

novamente, mas qual é o *layout* do teclado de seu computador. Se você conhece o modelo de seu teclado é fácil, pois basta selecionar, testar na barra acima do botão Detectar *layout* do teclado OS caracteres que normalmente apresentam erro, como acento, barra, barra invertida, cedilha, acentos e outros. Se as configurações funcionaram selecione Continuar para ir para a próxima tela, mas, em alguns casos, pode ser trabalhoso encontrar o teclado correto para seu computador. Não é recomendado você passar rápido para o próximo passo sem acertar o teclado mais compatível, visto que o Linux utiliza teclas especiais como ~/ e outros caracteres,



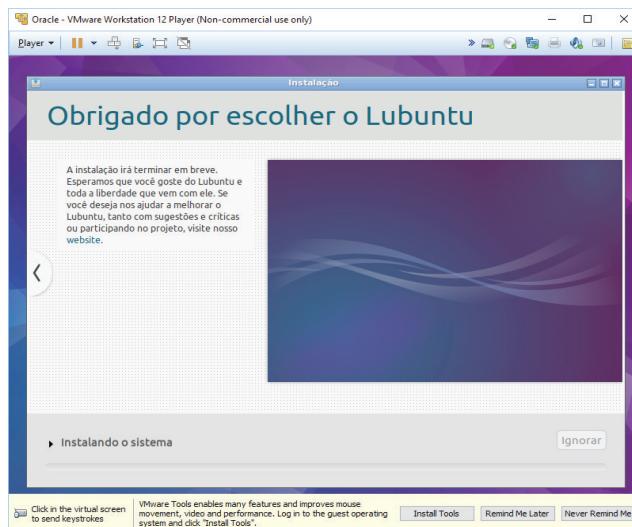
**Figura 31:** Definição do *layout* do teclado.

A figura 32 é apresentada após a seleção do *layout* do teclado. O Linux nesta tela deseja que você informe o seu nome, nome do computador, nome de um usuário e a senha do usuário. Para facilitar a aula, utilizei o nome oracle como meu nome, defini o nome do usuário como system e a senha como manager. Optei por iniciar a sessão automaticamente. Este usuário e senha são os padrões de instalação da Oracle. É claro que em um ambiente de produção estas escolhas não são recomendadas, mas no ambiente de desenvolvimento elas facilitam a equipe por não terem de memorizar duas senhas, uma para o Linux e outra para o Oracle. Você pode optar por alterar e personalizar essas informações, mas certifique-se de não perder essas informações.



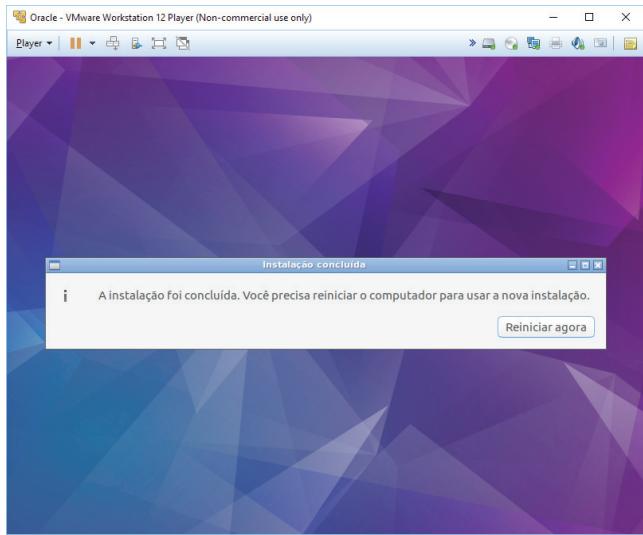
**Figura 32:** Identificação do usuário e senha.

Após definir os dados do usuário, o instalador inicia o processo de instalação e uma tela similar a apresentada na figura 33 é exibida.



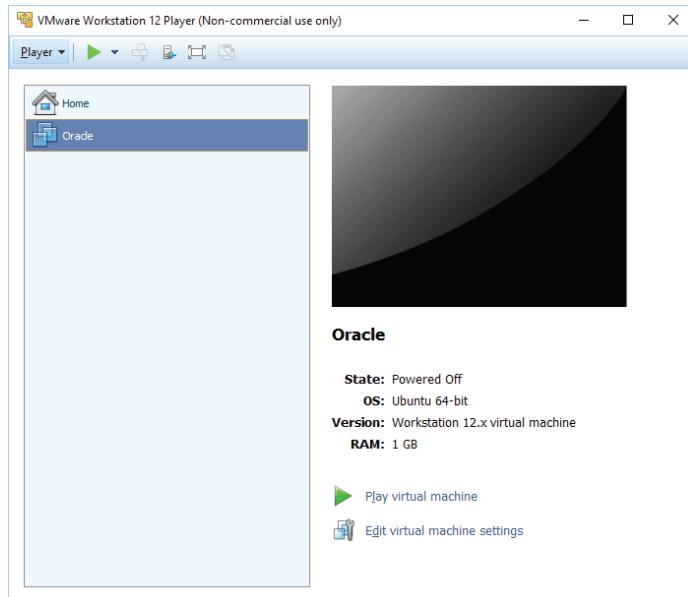
**Figura 33:** Processo de instalação do Linux Lubuntu.

A instalação demora um pouco e finaliza com a tela apresentada na figura 34 solicitando que se reinicie a VM.



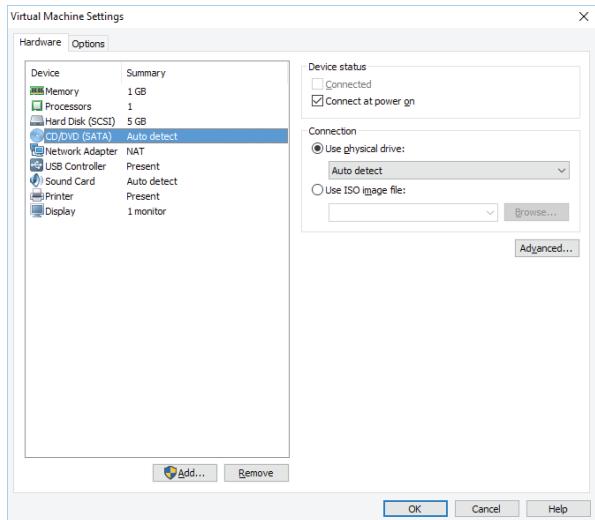
**Figura 34:** Solicitação para reiniciar a VM.

Após o computador desligar, interrompa o processo de reinicialização, pois você deve remover o disco de inicialização do sistema. Feche a VM e reabra como indica a figura 35.



**Figura 35:** VM com a VM Oracle criada.

Entre em *Edit virtual machine settings*. Selecione CD/DVD e mude a seleção de *Use ISO image file* ou use a imagem do arquivo ISO para *Use physical drive* ou use o *drive* físico.

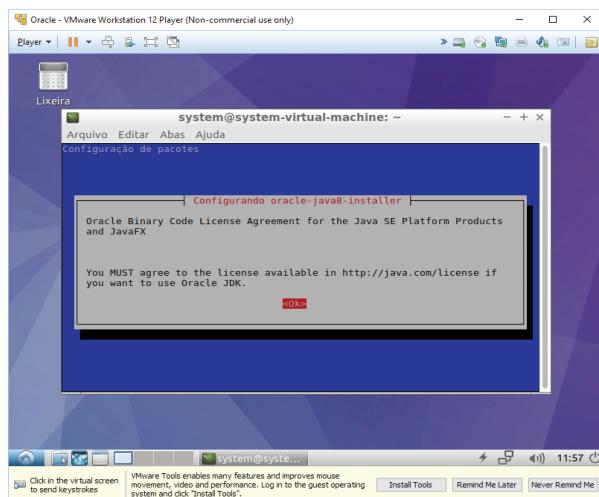


**Figura 36:** Modificando o boot da VM Oracle.

Reinicie o sistema e ação no Menu, Sistema *LXTerminal*. Digite na janela o comando representado na linha 1 e o sistema irá solicitar a senha do usuário *system*. Digite a senha definida, no meu caso é *manager*. As linhas 2 e 3 farão o sistema acrescentar um repositório, verificar e atualizar alguns arquivos. Após atualizar digite a linha 4, uma tela irá solicitar a confirmação da instalação.

1. sudo su
2. sudo add-apt-repository ppa:webupd8team/java
3. sudo apt-get update
4. sudo apt-get install oracle-java8-installer

Digite *ok* na próxima tela, como ilustrado na figura 37 e após *sim* ou *yes* para continuar a instalação do java8 na máquina virtual. Este processo deve demorar um pouco dependendo da velocidade de sua internet.



**Figura 37:** Instalando o Java 8 na VM.

Após a instalação, você deve confirmar que o Java está instalado corretamente, então execute o seguinte comando:

1. `java -version`

Espera-se um resultado parecido com o apresentado abaixo:

1. `java version "1.8.0_131"`

2. `Java (TM) SE RunTime Environment (build 1.8.0_131-b11)`

3. `Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)`

O próximo passo é configurar a variável de ambiente `JAVA_HOME`. Para fazer isso, abra o arquivo `bash.bashrc` executando a seguinte declaração.

1. `sudo apt install gedit`

2. `yes`

3. `sudo gedit /etc/bash.bashrc`

Desloque-se para a parte inferior do arquivo e adicione as seguintes linhas. É importante não alterar outras linhas do arquivo e não acrescentar espaço no texto.

1. `export JAVA_HOME=/usr/lib/jvm/java-8-oracle`

2. `export PATH=$JAVA_HOME/bin:$PATH`

Salve o arquivo selecionando o botão salvar na parte superior e feche o editor. Para carregar as alterações, execute a seguinte declaração.

1. `source /etc/bash.bashrc`

Para validar as alterações, você pode executar a seguinte declaração.

1. `echo $JAVA_HOME`

O resultado desta declaração deve ser o seguinte.

1. `/usr/lib/jvm/java-8-oracle`

Se a linha acima não aparecer, significa que você errou algum passo, refaça a alteração no arquivo `bash.bashrc` até que se tenha sucesso.

## ■ INSTALAR O SGBD ORACLE

Agora você deve instalar o SGBD *Oracle 11g Express Edition (XE)*. Para instalar o SGBD são necessários alguns pacotes adicionais, então digite o seguinte comando:

1. `sudo apt-get install alien libaiol unixodbc`

A instalação poderá pedir algumas autorizações e irá baixar alguns arquivos da internet. Autorize todas com yes e aguarde. Depois de finalizado, você deve abrir o navegador web para fazer o *download* do SGBD, um dos navegadores se encontra em Menu, Internet, Navegador Web Firefox. Digite o endereço <http://www.oracle.com/technetwork/products/express-edition/downloads/index.html> e selecione a versão Linux x64. Para baixar o arquivo você deverá selecionar que aceita os termos da licença e deve entrar com seu usuário na Oracle. Se você não tem um cadastro selecione *Sign In* e a opção *Create an account* e preencha o cadastro. Como apresentado na figura 38.

The screenshot shows the Oracle account creation interface. The form fields include:

- Email Address: claudineysanches@hotmail.com (validation message: Your email address is your username.)
- Password and Retype password: Both fields contain a placeholder password (validation message: Passwords must have upper and lower case letters, at least 1 number, not match any part of your email, and be at least 8 characters long.)
- Country: Brazil
- Name: Claudiyne Sanches (validation message: ✓)
- Job Title: Prof.
- Work Phone: +55 11-91234567 (validation message: ✓)
- Company Name: Faculdade das Americas
- Address: Rua da minha casa, 123 (validation message: ✓)
- City: São Paulo (validation message: ✓)
- State/Province: São Paulo
- ZIP/Postal Code: 04321-001 (validation message: ✓)

At the bottom of the form, there is a checkbox for "Yes, send me e-mails on Oracle Products, Services and Events." Below it, a note says "You may opt-out of all marketing communications: Unsubscribe." A small note also states: "By clicking on the 'Create Account' button below, you understand and agree that the use of Oracle's web site is subject to the Oracle.com Terms of Use and Oracle's Privacy Policy, including the fact that Oracle may transfer your personal information collected in connection with your registration on this website to its affiliates globally and to third party entities that provide services to Oracle."

**Figura 38:** Página de Cadastro de usuário da Oracle.

Após a conclusão do *Download*, abra o terminal e digite:

1. cd Downloads
  2. unzip oracle-xe-11.2.0-1.0.x86\_64.rpm.zip
  3. cd Disk1
  4. sudo alien --scripts -d oracle-xe-11.2.0-1.0.x86\_64.rpm
- Agora teremos que converter o padrão *rpm* que é da distribuição do *Linux Red Hat* para um pacote *Debian* com o comando *alien*. O parâme-

tro -d indica que um novo arquivo deve ser gerado e o parâmetro -script indica que ao instalar e remover, o *script* deve ser consultado. A execução pode demorar um pouco, não feche a janela. Abra uma nova janela com LXTerminal e enquanto aguardamos a conclusão da conversão, você pode antecipar alguns passos. No novo terminal digite:

1. sudo gedit /sbin/chkconfig

Esse comando vai criar o arquivo *chkconfig* que deverá ter as seguintes instruções:

```
#!/bin/bash
# Oracle 11gR2 XE installer chkconfig hack for Ubuntu
file=/etc/init.d/oracle-xe
if [[ ! 'tail -n1 $file | grep INIT' ]]; then
echo >> $file
echo '### BEGIN INIT INFO' >> $file
echo '# Provides: OracleXE' >> $file
echo '# Required-Start: $remote_fs $syslog' >> $file
echo '# Required-Stop: $remote_fs $syslog' >> $file
echo '# Default-Start: 2 3 4 5' >> $file
echo '# Default-Stop: 0 1 6' >> $file
echo '# Short-Description: Oracle 11g Express Edition' >> $file
echo '### END INIT INFO' >> $file
fi
update-rc.d oracle-xe defaults 80 01
#EOF
```

Salve as alterações no arquivo e feche a janela. Para alterar as permissões do arquivo que acabamos de criar digite:

1. sudo chmod 755 /sbin/chkconfig

O próximo passo é definir os parâmetros adicionais do *kernel* criando outro arquivo.

1. sudo gedit /etc/sysctl.d/60-oracle.conf

Digite os seguintes valores dentro do arquivo:

```
# Oracle 11g XE kernel parameters
fs.file-max=6815744
net.ipv4.ip_local_port_range=9000 65000
kernel.sem=250 32000 100 128
kernel.shmmax=536870912
```

Salve e feche o arquivo. Para verificar as alterações:

1. sudo cat /etc/sysctl.d/60-oracle.conf
2. sudo service procps start
3. sudo sysctl -q fs.file-max

Para fazer os últimos ajustes antes de instalar o SGBD digite:

1. sudo ln -s /usr/bin/awk /bin/awk
2. mkdir /var/lock/subsys
3. sudo touch /var/lock/subsys/listener

As alterações estão prontas. Verifique se, na outra janela, a conversão foi finalizada e o arquivo oracle-xe-11.2.0-2\_amd64.deb está gerado. Execute a instalação com o comando:

1. sudo dpkg --install oracle-xe\_11.2.0-2\_amd64.deb
2. sudo rm -rf /dev/shm
3. sudo mkdir /dev/shm
4. sudo mount -t tmpfs shmfs -o size=4096m /dev/shm

Pode ser que as linhas 2 e 3 não funcionem por estar o diretório em uso, neste caso passe para a linha 4. Assim, após a instalação, evitamos o erro de memória. Agora para finalizar, você deve criar o arquivo S01shm\_load, digite:

1. sudo gedit /etc/rc2.d/S01shm\_load

Dentro do editor digite os seguintes comandos:

```
#!/bin/sh
case "$1" in
start) mkdir /var/lock/subsys 2>/dev/null
touch /var/lock/subsys/listener
rm /dev/shm 2>/dev/null
mkdir /dev/shm 2>/dev/null
mount -t tmpfs shmfs -o size=4096m /dev/shm ;;
*) echo error
exit 1 ;;
esac
```

Salve e feche o arquivo. Agora temos de alterar o privilégio do arquivo que criamos. Digite:

1. sudo chmod 755 /etc/rc2.d/S01shm\_load

A instalação foi concluída, agora só falta fazer alguns ajustes para que o SGBD possa iniciar toda vez que o servidor for desligado ou reiniciado. Para isso você deve acrescentar algumas novas variáveis de ambiente no Linux. Abra o arquivo bash.bashrc com o comando:

1. sudo gedit /etc/bash.bashrc

Role com a seta para o final do arquivo e acrecente as seguintes linhas:

```
export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe
export ORACLE_SID=XE
export NLS_LANG='$ORACLE_HOME/bin/nls_lang.sh'
export ORACLE_BASE=/u01/app/oracle
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
export PATH=$ORACLE_HOME/bin:$PATH
```

Salve e feche o arquivo. Carregue as alterações com o comando:

1. source /etc/bash.bashrc

Para testar se as alterações foram aceitas digite:

```
1. echo $ORACLE_HOME
```

Você deverá ter como resposta a seguinte linha na tela:

```
/u01/app/oracle/product/11.2.0/xe
```

Agora para iniciar o banco de dados basta digitar a seguinte instrução:

```
1. sudo service oracle-xe start
```

Para criar um atalho para facilitar a tarefa de iniciar o banco de dados, digite:

```
1. cd ~/Área de Trabalho"
```

```
2. sudo gedit sgbd.desktop
```

Ao abrir o editor de texto com um arquivo novo, digite:

```
[Desktop Entry]
Exec=sudo service oracle-xe start
Terminal=true
MultipleArgs=true
Type=Applications;
Icon=oraclexe-startdatabase.png
MimeType=Application/database
Encoding=UTF-8
Name=Start Database
Name[pt_BR]=Iniciar Banco de Dados
Comment=Liga o SGBD
X-KeepTerminal=true
```

Agora você tem o servidor Linux com o SGBD Oracle pronto para utilizar. O SGBD Oracle é o com maior número de usuários nos últimos anos. Mas a interface não é a mais amigável, sendo recomendada uma ferramenta que auxilie na tarefa. A própria Oracle oferece a ferramenta SQL Developer gratuita que facilita muito a interação com o SGBD.

## ■ PARA INSTALAR SQLDEVELOPER.

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

Digite o comando:

```
1. cd Downloads
```

```
2. sudo alien --scripts -d sqldeveloper-4.2.0.17.089.1709-1.noarch.rpm
```

```
3. sudo dpkg --install sqldeveloper_4.2.0.17.089.1709-2_all.deb
```

4. sudo mkdir /home/.sqldeveloper/
5. sudo /opt/sqldeveloper/sqldeveloper.sh

A linha 1 nos leva a subpasta *downloads* onde o navegador salvou os arquivos baixados. A linha 2 converte a instalação rpm para a versão deb. A linha 3 inicia a instalação do SQL Developer. A linha 4 é a que inicia o SQL Developer, mas na primeira vez que rodar irá interromper para solicitar o caminho do Java. Assim digite:

```
/usr/lib/jvm/java-8-oracle
```

Após a ferramenta abrir e carregar no sistema, finalize pois você deverá criar um atalho para ela. Digite:

1. cd ~/Área de Trabalho"
2. sudo gedit sgldeveloper.desktop

Ao abrir o editor de texto com um arquivo novo, digite:

```
[Desktop Entry]
Encoding=UTF-8
Name=SQL Developer
Comment=Oracle SQL Developer
Icon=/opt/sqldeveloper/icon.png
Exec=sudo /opt/sqldeveloper/sqldeveloper.sh
Terminal=true
Type=Application
Path=/opt/sqldeveloper
Name [pt_BR]=SQL Developer
X-KeepTerminal=true
```



**síntese**

Nesta quarta aula, você aprendeu a instalar um servidor Linux. Viu como instalar o SGBD relacional Oracle em uma VM. Conheceu o processo de instalação da ferramenta de administração SQL Developer. Você pode notar os desafios para criar um ambiente de desenvolvimento, montando um servidor básico com o ferramental para utilizar nas próximas aulas.

## Atividade

01. No LXTerminal digite os seguintes comandos e tire os *print* das telas para comprovar que estão funcionando corretamente.

# AULA 5

## NORMALIZAÇÃO

### NESTA AULA

- » Formas normais
- » Ferramentas Case
- » Padronização no SGBD
- » Introdução à Linguagem SQL.

### I METAS DE COMPREENSÃO

- » Conhecer as etapas de Normalização.
- » Compreender o processo para colocar um projeto na forma Normal e quando desnormalizar. Conhecer algumas ferramentas case e aprender os padrões trigramação e o qualificador nome descritivo.

### I APRESENTAÇÃO

O objetivo desta aula é tratar das formas normais do projeto e da modelagem de um Banco de Dados Relacional. Você deve entender o processo de normalização e quando você deve criar um banco desnormalizado. Conhecer algumas ferramentas case o ajudará a se adaptar rápido a diferentes empresas. Também você notará que existem alguns padrões para Banco de Dados que podem lhe ajudar na modelagem.

Para entender estes conceitos, esta unidade está organizada da seguinte forma:

- » a seção 2 apresenta as formas normais de um banco de dados;
- » a seção 3 mostra as ferramentas case;
- » a seção 4 apresenta os conceitos da padronização trigramação e qualificador nome descritivo;
- » a seção 5 você vai iniciar a Linguagem SQL;

Não deixe de utilizar as bibliografias associadas à unidade.

Bom estudo!

# ■ FORMAS NORMAIS

## Normalização

processo para refinar e corrigir as estruturas de dados.

A técnica de **normalização** de um banco de dados permite mostrar que os bancos de dados relacionais apresentam vantagens no tratamento de dados. A normalização não permite que se crie redundância de dados, evita tratar os dados como o sistema de arquivos, evita criar anomalias e que se misture assuntos. A normalização otimiza a alocação de espaço, agrupa dados de forma que cada dado seja dependente de sua chave primária, prove um meio formal de se estruturar a informação e ajuda a diluir dúvidas do projeto. As técnicas de levantamento de requisito, análise e modelagem vistas até esta aula permitem criar uma boa estrutura de dados, mas às vezes, pode-se criar uma estrutura ruim de dados. Para reconhecer e classificar as estruturas de dados, existe a normalização. Normalização é um processo para refinar e corrigir as estruturas de dados propostas que atua por meio de uma série de estágios chamados **Formas Normais** (FN). Das cinco formas normais disponíveis, os três primeiros estágios, para a maioria dos projetos comerciais são suficientes. Vale ressaltar que quanto maior a normalização, menos espaço em HD o banco vai ocupar e maior será a demanda de processamento, uniões, relacionamentos etc. Embora a normalização seja importante no banco de dados, não presuma que todos os projetos devem atingir o maior nível de normalização pois você deve avaliar a demanda de desempenho e em alguns casos desnortinalizar um banco de dados para uma Forma Normal (FN) mais baixa. O processo de normalização causa a simplificação de atributos de uma entidade, tornando o banco de dados mais estável e de fácil manutenção.

## Formas Normais

estágios que atuam para a Normalização do banco de dados.

## ETAPA 1: NA 1<sup>a</sup> FN VOCÊ DEVE ELIMINAR OS GRUPOS DE REPETIÇÃO

Ao apresentar os dados em formato de tabela, elimine os valores nulos e separe os grupos de repetição. Para facilitar o entendimento veja a figura 01 com suas tuplas, representando o relatório de matrículas da Faculdade. Observe que existem colunas agrupadas, indicando a existência de dados que se repetem. Neste caso, a ideia é que a partir do relatório, você consiga modelar as entidades e os atributos de maneira correta.

## Relatório de Matrículas

nome_aluno	rgm	nome_pai	nome_mae	nascimento	sexo	data_matricula	semestre	turma	disciplinas
Julia Izabel Ribeiro	25100	Luciano Alves Ribeiro	Josiane Ribeiro	25/05/1995	F	10/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Banco de Dados Sistemas Operacionais Redes Neurais
Amanda Esposito	25101	Edson Franco Esposito	Luciana Antunes Espotito	24/02/1996	F	12/01/2017	2º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial
Roberto Rodrigo Prado	25102	Carlos Eduardo Prado	Eliana Pinto Prado	15/01/1994	M	10/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial
Fernando Franchini	25103	Roberto Serafin Franchini	Daniela Andrade Franchini	10/02/1990	F	11/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial
Nelson Martins	25104	Durval Pilon Martins	Clemilda Santos Martins	16/02/1992	M	09/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial
Raphael Lucas Fernandes	25105	Wendel Garcia Fernandes	Andiara Rodrigues Fernandes	30/10/1993	M	18/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial
Adriana Bonfieti Moraes	25106	Ewerton Pereira Moraes	Elena Sagio Moraes	05/05/1994	F	10/01/2017	1º	A	Lógica e Técnicas de Programação Arquitetura de Computadores Introdução à Inteligência Artificial Banco de Dados Sistemas Operacionais Redes Neurais
Berenice Amaral Martins	25107	Pedro Santos Pereira	Gabriela Aparecida Pereira	01/06/1995	F	09/01/2017	2º	A	Banco de Dados Sistemas Operacionais Redes Neurais
Alan Jacinto Patrício	25108	Franklin Bitencourt Patrício	Nathalia França Patrício	27/08/1996	M	15/01/2017	2º	A	Sistemas Operacionais Redes Neurais

**Figura 01:** Exemplo de entidade não normalizada.

A primeira forma normal exige que se elimine as repetições, para isso, você deve dividir a tabela em duas tabelas, como indicado na figura 2. Além disso, todo o atributo da tabela deve ser atômico ou indivisível. Não é permitido atributo multivalorado, composto ou multivalorado composto. Nesta etapa o esquema fica assim:

**Aluno(rgm, nome\_aluno, nome\_pai, nome\_mae, nascimento, sexo, data\_matricula, semestre, turma, id\_semestre)**

**Disciplina(id\_semestre, disciplinas)**

A regra da 1ª FN: se uma entidade apresentar grupos de repetição, desmembrá-la criando uma nova entidade.

Aluno	nome_aluno	rgm	nome_pai	nome_mae	nascimento	sexo	data_matricula	semestre	turma	id_semestre
Julia Izabel Ribeiro	25100	Luciano Alves Ribeiro	Josiane Ribeiro	25/05/1995	F	10/01/2017	1º	A		1
Amanda Esposito	25101	Edson Franco Esposito	Luciana Antunes Espotito	24/02/1996	F	12/01/2017	2º	A		2
Roberto Rodrigo Prado	25102	Carlos Eduardo Prado	Eliana Pinto Prado	15/01/1994	M	10/01/2017	1º	A		1
Fernando Franchini	25103	Roberto Serafin Franchini	Daniela Andrade Franchini	10/02/1990	F	11/01/2017	1º	A		1
Nelson Martins	25104	Durval Pilon Martins	Clemilda Santos Martins	16/02/1992	M	09/01/2017	1º	A		1
Raphael Lucas Fernandes	25105	Wendel Garcia Fernandes	Andiara Rodrigues Fernandes	30/10/1993	M	18/01/2017	1º	A		1
Adriana Bonfieti Moraes	25106	Ewerton Pereira Moraes	Elena Sagio Moraes	05/05/1994	F	10/01/2017	1º	A		1
Berenice Amaral Martins	25107	Pedro Santos Pereira	Gabriela Aparecida Pereira	01/06/1995	F	09/01/2017	2º	A		2
Alan Jacinto Patrício	25108	Franklin Bitencourt Patrício	Nathalia França Patrício	27/08/1996	M	15/01/2017	2º	A		2

### Disciplina

id_semestre	disciplinas
1	Lógica e Técnicas de Programação
1	Arquitetura de Computadores
1	Introdução à Inteligência Artificial
2	Banco de Dados
2	Sistemas Operacionais
2	Redes Neurais

**Figura 02:** 1ª FN sem grupos de repetição.

## ETAPA 2: NA 2<sup>a</sup> FN DEVE EXISTIR APENAS UMA CHAVE PRIMÁRIA.

Para colocar as entidades na 2<sup>a</sup> FN, você deve garantir que elas estejam na 1<sup>a</sup> FN e que cada coluna, que não seja chave primária, dependa exclusivamente da chave primária. O objetivo será corrigir a tabela, fazendo com que os assuntos sejam separados, assim se um atributo depender apenas de uma parte da chave primária, você deve desmembrá-lo criando uma nova entidade em que todos os atributos dependam da chave primária. Nesta etapa, o esquema fica assim:

**Aluno(rgm, nome\_aluno, nome\_pai, nome\_mae, data\_nascimento, sexo)**  
**Matricula(rgm, data\_matricula, id\_turma, id\_semestre)**  
**Disciplina(id\_disciplina, ds\_disciplinas, id\_semestre)**  
**Semestre(id\_semestre, ds\_semestre)**  
**Turma(id\_turma, ds\_turma)**

Observe por exemplo a figura 02, a tabela Aluno apresenta como chave primária o rgm e as colunas nome\_aluno, nome\_pai, nome\_mae, nascimento e sexo são dependentes da chave primária. As demais colunas não são dependentes e devem ser desmembradas. Quanto à segunda tabela, Disciplina não apresenta chave primária então você deve acrescentar a chave primária. A figura 3 apresenta as correções para a 2<sup>a</sup> FN.

Aluno					Disciplina			
rgm	nome_aluno	nome_pai	nome_mae	data_nascimento	sexo	id_disciplina	ds_disciplinas	id_semestre
25100	Julia Izabel Ribeiro	Luciano Alves Ribeiro	Josiane Ribeiro	25/05/1995	F	1	Lógica e Técnicas de Programação	1
25101	Amanda Esposito	Edson Franco Esposito	Luciana Antunes Espotito	24/02/1996	F	2	Arquitetura de Computadores	1
25102	Roberto Rodrigo Prado	Carlos Eduardo Prado	Eliana Pilot Prado	15/01/1994	M	3	Introdução a Inteligência Artificial	1
25103	Fernando Franchini	Roberto Serafin Franchini	Daniela Andrade Franchini	10/02/1990	F	4	Banco de Dados	2
25104	Nerson Martins	Durval Pilon Martins	Clemildes Santos Martins	16/02/1992	M	5	Sistemas Operacionais	2
25105	Raphael Lucas Fernandes	Wendel Garcia Fernandes	Andriara Rodrigues Fernandes	30/10/1993	M	6	Redes Neurais	2
25106	Adriana Bonfetti Moraes	Ewerton Pereira Moraes	Elena Sagio Moraes	05/05/1994	F			
25107	Berenice Amaral Martins	Durval Pilon Martins	Clemildes Santos Martins	01/06/1995	F			
25108	Alan Jacinto Patrício	Franklin Bitencourt Patrício	Nathalia França Patrício	27/08/1996	M			
						Semestre		
						id_semestre	ds_semestre	
						1	1º Básico de Programação	
						2	2º Banco de Dados	
						3	3º Análise de Sistemas	
						4	4º Gestão de Projetos	
						5	5º Conclusão	
						Turma		
						id_turma	ds_turma	
						1	A - Noturno	
						2	B - Noturno	
						3	A - Diurno	
						4	B - Diurno	

Matricula			
rgm	data_matricula	id_turma	id_semestre
25100	10/01/2017	1	1
25101	12/01/2017	1	2
25102	10/01/2017	1	1
25103	11/01/2017	1	1
25104	09/01/2017	1	1
25105	18/01/2017	1	1
25106	10/01/2017	1	1
25107	09/01/2017	1	2
25108	15/01/2017	1	2

**Figura 03:** 2<sup>a</sup> FN campos dependentes da chave primária.

**Etapa 3: Na 3<sup>a</sup> FN identificar atributos que não são dependentes das chaves primárias.**

Para colocar as entidades na 3<sup>a</sup> FN, você deve garantir que esteja na 2<sup>a</sup> FN e que cada atributo seja dependente da chave primária ou de parte dela. Se algum atributo for dependente de outro campo, estes devem ser desmembrados em uma nova entidade. O objetivo é corrigir a presença de atributos dependentes de outros campos que não sejam chave primária. Além disso, se existir campos calculados, esses devem ser eliminados. Nesta etapa o esquema fica assim:

**Aluno(rgm, nome\_aluno, nome\_pai, nome\_mae, data\_nascimento, sexo)**  
**Matricula(rgm, data\_matricula, id\_classe)**  
**Classe(id\_classe, id\_turma, id\_semestre)**  
**Disciplina(id\_disciplina, ds\_disciplinas, id\_semestre)**  
**Semestre(id\_semestre, ds\_semestre)**  
**Turma(id\_turma, ds\_turma, ano\_turma)**

No exemplo apresentado na figura 03, a entidade matricula apresenta dois atributos que não dependem da chave primária rgm. A figura 04 apresenta a correção, dividindo a entidade em duas entidades, Matricula e Classe, assim consegue-se organizar de forma que todos os atributos dependam das chaves primárias de suas entidades, colocando o esquema na 3<sup>a</sup> FN.

Aluno					
rgm	nome_aluno	nome_pai	nome_mae	data_nascimento	sexo
25100	Julia Izabel Ribeiro	Luciano Alves Ribeiro	Josiane Ribeiro	25/05/1995	F
25101	Amanda Esposito	Edson Franco Esposito	Luciana Antunes Espotito	24/02/1996	F
25102	Roberto Rodrigo Prado	Carlos Eduardo Prado	Eliana Pinto Prado	15/01/1994	M
25103	Fernando Franchini	Roberto Serafin Franchini	Daniela Andrade Franchini	10/02/1990	F
25104	Nerson Martins	Durval Pilon Martins	Clemílides Santos Martins	16/02/1992	M
25105	Raphael Lucas Fernandes	Wendel Garcia Fernandes	Andiara Rodrigues Fernandes	30/10/1993	M
25106	Adriana Bonfietti Moraes	Everton Pereira Moraes	Elena Sagio Moraes	05/05/1994	F
25107	Berenice Amaral Martins	Durval Pilon Martins	Clemílides Santos Martins	01/06/1995	F
25108	Alan Jacinto Patrício	Franklin Bitencourt Patrício	Nathalia França Patrício	27/08/1996	M

Matricula					
rgm	data_matricula	id_classe	id_turma	ds_turma	ano_turma
25100	10/01/2017	1	1	A - Noturno	2016
25101	12/01/2017	2	2	B - Noturno	2017
25102	10/01/2017	1	3	A - Diurno	2016
25103	11/01/2017	1	4	B - Diurno	2017
25104	09/01/2017	1			
25105	18/01/2017	1			
25106	10/01/2017	1			
25107	09/01/2017	2			
25108	15/01/2017	2			

Disciplina					
id_disciplina	ds_disciplina	id_semestre	id_turma	ds_turma	ano_turma
1	Lógica e Técnicas de Programação	1	1	A - Noturno	2016
2	Arquitetura de Computadores	1	2	B - Noturno	2017
3	Introdução à Inteligência Artificial	1	3	A - Diurno	2016
4	Banco de Dados	2	4	B - Diurno	2017
5	Sistemas Operacionais	2	5	Conclusão	
6	Redes Neurais	2			

Semestre					
id_semestre	ds_semestre	id_turma	ds_turma	ano_turma	
1	1º Básico de Programação	1	A - Noturno	2016	
2	2º Banco de Dados	2	B - Noturno	2017	
3	3º Análise de Sistemas	3	A - Diurno	2016	
4	4º Gestão de Projetos	4	B - Diurno	2017	
5	5º Conclusão	5			

Turma					
id_turma	ds_turma	ano_turma	id_classe	id_turma	ds_turma
1	A - Noturno	2016	1	1	
2	B - Noturno	2017	2	2	
3	A - Diurno	2016			
4	B - Diurno	2017			

Classe					
id_classe	id_turma	ds_turma	ano_turma	id_semestre	
1	1	A - Noturno	2016	1	
2	2	B - Noturno	2017	2	

**Figura 04:** 3<sup>a</sup> FN atributos dependendo da chave primária.

## DEMAIS FORMAS NORMAIS.

Para a maioria das análises de requisitos e projetos de decomposição, as 3 etapas iniciais são suficientes para obter um banco de dados rápido

e eficiente. Na literatura existe outras formas normais, como a forma normal de Boyce/Codd ou BCNF, a 4<sup>a</sup> FN e 5<sup>a</sup> FN. A 4<sup>a</sup> FN é empregada se, após a 3<sup>a</sup> FN existir redundância, como a utilização de atributos de valores múltiplos para a mesma chave primária. A 5<sup>a</sup> FN é um caso muito raro de ocorrer, como por exemplo chegar a 4<sup>a</sup> FN com uma entidade com 3 atributos multivalorados. A forma normal BCNF aperfeiçoa a 3<sup>a</sup> FN à medida que faz análise das chaves candidatas, impondo que só exista atributos dependentes da chave primária ou da chave candidata. Se existir atributos fora desta especificação, deve ser separado em uma nova entidade.

## DESNORMALIZAÇÃO

A implementação ideal de um banco de dados relacional exige que todas as entidades estejam pelo menos na 3<sup>a</sup> FN. Um bom banco de dados não apresenta redundâncias desnecessárias que podem causar anomalias. Embora normalizar torne o armazenamento mais eficiente, visto que os dados ficam no menor tamanho possível, exige maior quantidade de processamento nas consultas à base de dados. Assim, ao projetar um banco de dados você deve também analisar a necessidade de velocidade de processamento e pensar na questão da *performance* x demanda de usuários. O problema encontrado com as FN é que ao aplicá-las, notamos que a quantidade de entidades geradas aumenta e para gerar informações úteis aos usuários será necessário unir os dados que estão espalhados em diversas tabelas. A junção de um grande número de tabelas vai exigir processamento do servidor. O número de operações de entrada e saída aumenta e com isso aumenta o tempo para o servidor responder uma solicitação. A maioria dos SGBDs comerciais é capaz de tratar um grande número de junções, mas em alguns casos, a quantidade de acessos simultâneos ao servidor pode gerar um atraso significativo no tempo de resposta. Tenha bom senso ao avaliar até que ponto se deve normalizar um banco, reduzindo o tamanho e a possibilidade de anomalias, ou desnormalizar aumentando o desempenho e podendo apresentar redundância de dados e anomalias em resultados. Um bom projeto de banco de dados não pode ter uma visão única de *performance*, pois os dados serão utilizados ao longo do tempo e é bem provável que sejam mais perenes do que uma aplicação. Tenha em mente que a desnormalização deve ser utilizada com cuidado sempre com uma explicação de necessidade de tal ação.

# FERRAMENTAS CASE

A ferramenta do tipo *CASE - Computer Aided Software Engineering*, ou seja, ferramentas de engenharia de software auxiliada por computador são projetadas para lhe auxiliar no processo de engenharia de software. Existem no mercado diversas ferramentas, com foco amplo ou específico, cuja finalidade é auxiliar o engenheiro de software. Nesta aula você deve focar apenas nas ferramentas que podem auxiliar na área de Banco de Dados. Algumas ferramentas de interesse são *DBDesigner*, *Erwin*, *Oracle Designer*, *brModelo* e *Workbench*. É claro que essas são apenas algumas das ferramentas existentes, mas já conseguem transmitir o conceito de **ferramenta CASE** e permitir explorar mais ferramentas existentes.

As principais características esperadas de ferramentas CASE para banco de dados são que elas suportem criar diagramas DER, suportem escrever comandos **SQL**, **forward engineer**, **reverse engineer** e acompanhamento da documentação. *Forward engineer* é quando a partir de um DER você consegue criar automaticamente o modelo físico no banco de dados. Este recurso permite que os modelos conceitual e lógico sejam criados de forma automática com a possibilidade de edição em cada etapa. *Reverse engineer* é quando a partir de um banco de dados físico, já implementado, a ferramenta consegue criar para você o diagrama DER automaticamente. Normalmente este recurso é utilizado quando um banco de dados não apresenta em sua documentação o diagrama, quer por ser parte de software legado quer por ser parte de um processo sem controle. A documentação deve apresentar suporte para o dicionário de dados e permitir acompanhar a evolução do banco de dados. As vantagens das ferramentas CASE são: maior velocidade no desenvolvimento de projetos, melhor qualidade dos processos, melhor documentação visto que integra DER, banco de dados físico e documentação e uso de interface gráfica. Com exceção da CA *Erwin* todos as demais ferramentas indicadas na tabela 01 são gratuitas. A tabela 1 apresenta os sites onde você pode encontrar as ferramentas para baixar. Vale lembrar que *Workbench* é compatível com o MySQL e não cria facilidade para ser utilizada com o *Oracle*. O *DBDesigner* permi-

## Ferramenta CASE

ferramentas de engenharia de software que são projetadas para lhe auxiliar.

## SQL

Structured Query Language ou linguagem estruturada de consulta.

## Forward engineer

a partir de um DER consegue-se criar automaticamente o modelo físico.

## Reverse engineer

a partir de um banco de dados físico, já implementado, a ferramenta consegue criar o diagrama DER automaticamente.

te a compatibilidade com o *Oracle*, enquanto *Erwin*, *brModelo* e *Oracle Developer DataModeler* são compatíveis com o SGBD *Oracle*.

**Tabela 01:** Ferramentas Case e endereços para download.

NOME	SITE
Erwin	<a href="http://erwin.com/products/data-modeler/">http://erwin.com/products/data-modeler/</a>
DBDesigner	<a href="https://dbdesigner.br.uptodown.com/windows">https://dbdesigner.br.uptodown.com/windows</a>
brModelo	<a href="http://sis4.com/brModelo/brModelo/download.html">http://sis4.com/brModelo/brModelo/download.html</a>
Oracle Developer DataModeler	<a href="http://www.oracle.com/technetwork/developer-tools/datamodeler/downloads/datamodeler-087275.html">http://www.oracle.com/technetwork/developer-tools/datamodeler/downloads/datamodeler-087275.html</a>
Workbench	<a href="https://dev.mysql.com/downloads/workbench/">https://dev.mysql.com/downloads/workbench/</a>

## ■ PADRONIZAÇÃO NO SGBD

Não existe um consenso sobre como se deve criar o projeto físico do SGBD nem sobre quais regras de nomenclatura devem ser adotadas para tabelas e colunas. A padronização de nomes, campos, tabelas é importante pois facilita o entendimento e a velocidade de desenvolvimento de analistas e programadores. Quando você está modelando e trabalhando com um banco de dados pequeno, pode parecer perda de tempo discutir e definir uma política de padronização, mas quando o número de tabelas é grande as coisas mudam e pode se tornar muito difícil extrair informações de um banco não padronizado. Existem dois padrões que inspiram muitos DBA a criar suas regras que são variações para cada SGBD. A **trigramação** e o **qualificador nome descritivo**.

### Trigramação

cadeia de caracteres, normalmente constituída por três letras da entidade.

### Qualificador nome descritivo

todas as entidades e atributos do DER devem de alguma forma ser representados por símbolos contido nas tabelas de símbolos.

## TRIGRAMAÇÃO.

A trigramação é uma cadeia de caracteres, normalmente constituída por três letras da entidade ou pela escolha das três letras mais significativas de uma entidade. Essas três letras são utilizadas como parte da definição do atributo da mesma entidade. Para exemplificar a figura 05 apresenta a entidade Cliente e ela convertida para trigramação.



**Figura 05:** Trigramação.

Como regra geral da trigramação, nunca utilize artigo como ‘de’, ‘das’ ou ‘com’ ou pronomes em uma entidade ou atributo. Não utilize caracteres especiais, acentos ou cedilha. Todas as palavras compostas devem ser separadas por ‘\_’ underscore como pessoa\_juridica ou pessoa\_fisica. O nome da entidade deve ser sempre no singular e deve expressar de forma clara a finalidade da entidade. Na trigramação, ao se criarem abreviaturas, pode ocorrer de ao escolher as três primeiras letras de uma entidade, encontrar outra entidade que apresente as três letras iniciais idênticas. Neste caso, você deve manter duas letras idênticas e escolher uma terceira que permita diferenciar as entidades. Se o modelo for complexo e tiver a necessidade de se criar módulos, o nome do módulo será abreviado e atribuído a todas as tabelas que lhe pertence. Toda a chave primária terá a trigramação seguido da palavra id.

## QUALIFICADOR NOME DESCRIPTIVO.

O qualificador nome descritivo tem como base o padrão ISO/IEC 11179-5. A regra utiliza basicamente algumas tabelas com símbolos e todas as entidades e atributos do DER devem, de alguma forma, ser representados por um destes símbolos. Por exemplo, imagine que queremos converter a entidade Cliente do DER para o padrão qualificador nome descritivo. Você deverá procurar para cada atributo da entidade na tabela 02 o símbolo mais adequado. A figura 06 apresenta como fica a representação da entidade no modelo.



**Figura 06:** Qualificador nome descritivo.

A tabela 02 apresenta todos os qualificadores definidos com seus respectivos significados. Quando você for utilizar o qualificador nome descritivo converta o atributo do DER para um dos qualificadores apresentado na tabela 02. Faça isso para todos os atributos das entidades.

**Tabela 02:** Qualificador para Atributos.

QUALIFICADOR	SIGNIFICADO
cd_+[nome da coluna]	Código.
no/nm_+[nome da coluna]	Nome.
nr_+[nome da coluna]	Número.
vl_+[nome da coluna]	Valor.
qt_+[nome da coluna]	Quantidade.
tx_+[nome da coluna]	Taxa ou percentual.
ds_+[nome da coluna]	Descrição.
sg_+[nome da coluna]	Sigla.
dt_+[nome da coluna]	Data.
hr_+[nome da coluna]	Hora.
id/ie_+[nome da coluna]	Identificador.
im_+[nome da coluna]	Imagem.
st_+[nome da coluna]	Situação ou status.
cg_+[nome da coluna]	Coordenadas Geográficas.
au_+[nome da coluna]	Auditória.

Quanto ao nome da própria entidade, você deve achar um qualificador mais adequado utilizando a tabela 03 que apresenta os qualificadores que podem ser utilizados e seus significados.



você sabia?

O DATASUS definiu uma metodologia para a administração de banco de dados que tem como base alguns estudos feitos sobre a norma ISO/IEC 11179-5. Você pode ler a norma criada e utilizar o resumo para adequar a norma para o seu banco de dados. Acesse:

<http://datasus.saude.gov.br/estrutura-mad/norma-mad-menu> e

[http://datasus.saude.gov.br/images/MAD/files/DAAED\\_MAD\\_GuiaModelagemDados-221.pdf](http://datasus.saude.gov.br/images/MAD/files/DAAED_MAD_GuiaModelagemDados-221.pdf)

**Tabela 03:** Qualificador para Entidades.

QUALIFICADOR	SIGNIFICADO
tb_+[nome da tabela]	Tabela do software.
rl_+[nm_tb_pai]_[nm_tb_filho]	Tabela de Relacionamento.
au_+[nome da tabela]	Tabela de Auditoria.
tm_+[nome da tabela]	Tabela Temporária.
th_+[nome da tabela]	Tabela de Histórico.
ta_+[nome da tabela]	Tabela Auxiliar.
bk_+[nome da tabela]	Tabela de Backup.
rt_+[nome da tabela]	Tabela Relacionamento Ternário.

É importante lembrar que, como não existe um consenso, você está livre para criar seu padrão para o banco de dados. Ele deve permitir que o banco cresça e suporte ter milhares de tabelas, sendo fácil de usar, evitando ambiguidades, permitindo acessar o dado de maneira correta.

## ■ INTRODUÇÃO À LINGUAGEM SQL

SQL é a linguagem de interface para os SGBDs relacionais, assim, todos os usuários que querem interagir com o banco de dados deve fornecer comandos escritos nela. A linguagem SQL foi desenvolvida no início da década de 70 pela *IBM Research* no laboratório em San Jose com o nome SEQUEL, tendo seu nome modificado mais tarde. SQL significa *Structured Query Language* ou linguagem estruturada de consulta. O SQL permite a criação de banco de dados, executar tarefas de gerenciamento, consultar, transformar dados, executar funções e gatilhos. Além disso, deve ser padronizado para que o usuário não tenha que reaprender o básico quando passar de um SGBD para outro; assim, em 1987, o *American National Standard Institute* **ANSI** publicou o padrão SQL. A linguagem foi concebida para ser usada segundo o modelo de Codd e com o tempo ganhou outras funcionalidades.

O nascimento do SQL se deu devido à necessidade de trabalhar de forma fácil e flexível com junções, produto cartesiano, diferença, intersecção, projeção e união de dados. Praticamente todos os fornecedores de SGBD relacionais oferecem o SQL como *interface*. Suas funções se enquadram em duas categorias: Linguagem de definição de dados (**DDL**) e Linguagem de manipulação de dados (**DML**).

A DDL inclui comandos para criar objetos de banco de dados como tabelas, índices e visualizações, bem como comandos para definir direitos de acesso. Alguns comandos de definição de dados que veremos nesta aula são apresentados na tabela 4.

### ANSI

American National Standard Institute.

### DDL

Linguagem de definição de dados.

### DML

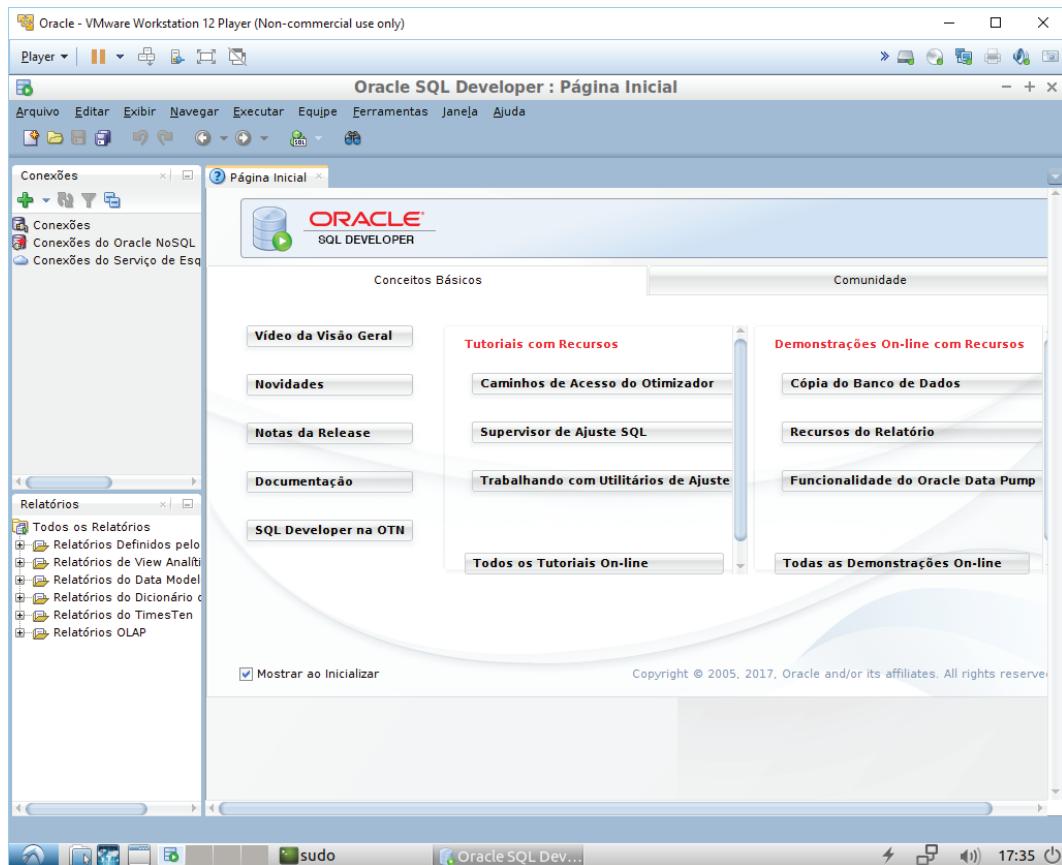
Linguagem de manipulação de dados.

**Tabela 04:** Alguns comandos DDL.

COMANDO	DESCRIÇÃO
CREATE SCHEMA AUTHORIZATION	Cria um esquema de banco de dados.
CREATE TABLE	Cria uma nova tabela.
NOT NULL	A coluna não poderá ter valores nulos.
UNIQUE	O valor será único na coluna.
PRIMARY KEY	Define a coluna como chave primária.
FOREING KEY	Define a coluna como chave estrangeira.
DEFAULT	Define um valor padrão para a coluna.
CHECK	Valida os dados de um atributo.
CREATE INDEX	Cria um índice para uma tabela.
CREATE VIEW	Cria um subconjunto dinâmico linhas e colunas a partir de uma ou mais tabelas .
ALTER TABLE	Modifica a definição de uma tabela.
CREATE TABLE AS	Cria um tabela com base em uma consulta.
DROP TABLE	Exclui uma tabela e seus dados.
DROP INDEX	Exclui um índice.
DROP VIEW	Exclui uma visualização.

Você vai gostar de saber que aprender o SQL é relativamente fácil. Você terá de aprender aproximadamente 100 instruções. É um vocabulário simples e que por ser padrão ANSI/ISO significa que aprendendo SQL você conseguirá migrar para diferentes SGBDRs. Mas cada fabricante adicionou recursos que dificultam a migração de uma aplicação que foi desenvolvida em um SGBDR para outro. Assim, você notará que aprender SQL é igual a falar um idioma, dependendo da região, surgem expressões e dialetos que podem dificultar a comunicação em alguns momentos. Mas, para este caso, uma boa consulta no manual ou na internet pode sanar rapidamente as dificuldades.

Antes de iniciar a utilização do SQL, você deve conhecer a ferramenta *SQL Developer* que foi instalada na aula anterior. A figura 07 apresenta a interface do *SQL Developer*.

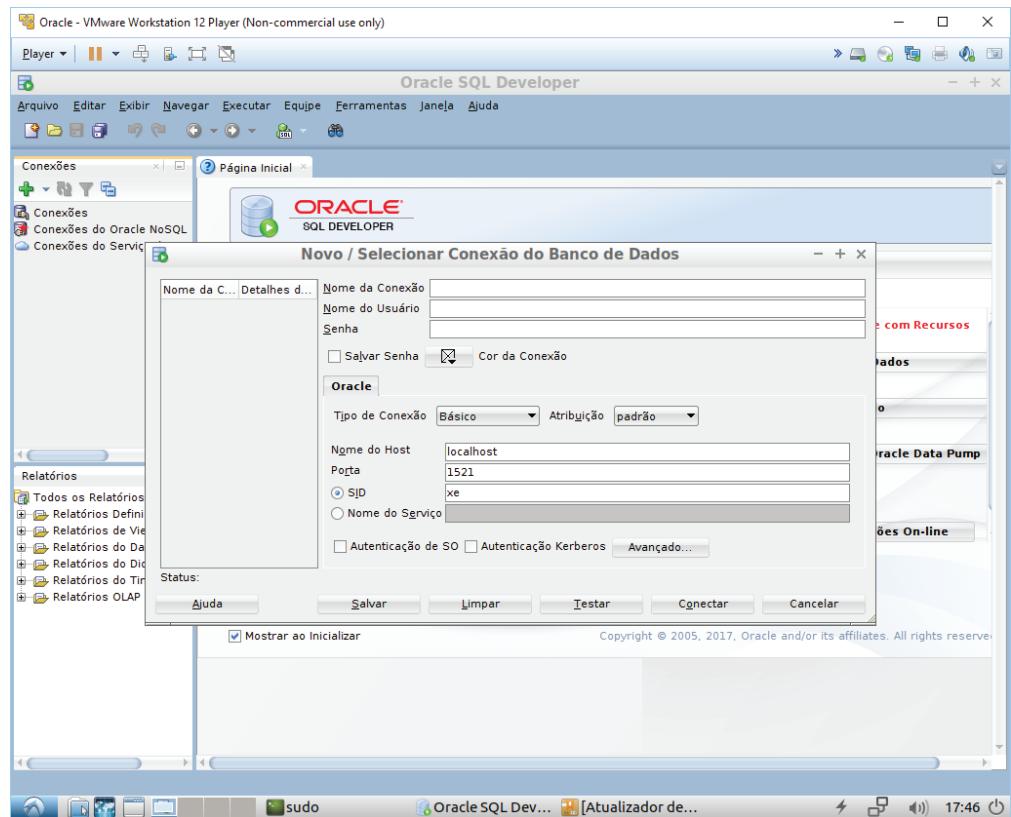


**Figura 07:** Tela do SQL Developer.

Selecione Arquivo, Novo, Camada do Banco de Dados e Botão Ok ou, se preferir pode utilizar o atalho <Ctrl><N> pressionando as teclas simultaneamente. Ainda existe a possibilidade de executar o mesmo comando clicando sobre o + verde que aparece na interface gráfica na janela conexões. Se optar por esta ação, selecione Nova Conexão e uma janela semelhante à apresentada na figura 08 deverá aparecer.

Você deve criar uma conexão para o DBA do SGBD e uma segunda conexão para utilizar nesta aula. Observe que a conexão DBA não deve ser utilizada como sua conexão particular, pois você deve deixar este super usuário para criar outros usuários, definir políticas de acesso, *tuning* de banco de dados e outras atividades administrativas do banco. Por isso, nesta aula você irá criar duas conexões: a DBA e Aluno.

Para criar o usuário DBA no campo nome, digite “DBA – Oracle”, no campo nome do usuário digite “system” e em senha digite “manager” conforme definimos no processo de instalação. Selecione o campo “Salvar Senha” e selecione o botão Testar. Se aparecer no Status: “Sucesso” selecione o botão Conectar.



**Figura 08:** Nova Conexão do SQL Developer com o SGBD Oracle.

Observe que a nova conexão foi criada. Selecione o + que aparece antes da Conexão DBA – Oracle e a interface deve ficar semelhante à indicada na figura 09.

Observe que, na janela à direita, uma nova aba foi criada com o nome da conexão e um espaço para enviarmos comandos SQL para o SGBD. Para criar um usuário no Oracle você deve utilizar a instrução *CREATE USER*. De maneira simplificada, o comando tem a seguinte sintaxe:

```
CREATE USER nome_do_usuario IDENTIFIELD BY senha_usuario
DEFAULT TABLESPACE nome_tablespace
TEMPORARY TABLESPACE tablespace_temporaria;
```

Onde

nome\_do\_usuario: é o nome do usuário que você quer criar;

senha\_usuario: é a senha que este usuário terá;

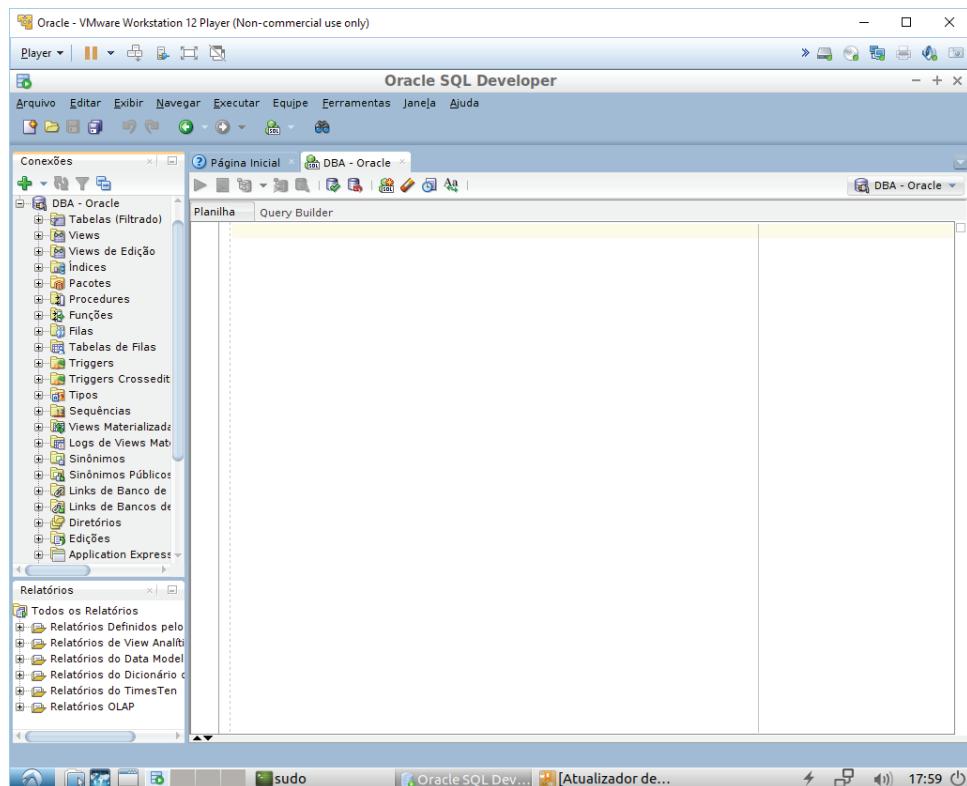
nome\_tablespace: onde os objetos do usuário serão armazenados;

tablespace\_temporaria: onde são armazenados os dados temporários do usuário, como por exemplo uma tabela temporária.

Então digite o seguinte comando:

```
CREATE USER aluno IDENTIFIELD BY aluno
DEFAULT TABLESPACE users
```

```
TEMPORARY TABLESPACE temp;
```



**Figura 09:** Conexão DBA – Oracle criada.

Para executar o comando, você pode pressionar as teclas <Ctrl><Enter> estando o cursor sobre a linha de comando ou selecionar o ícone play abaixo do título da aba. Veja na figura 10 o ícone de execução dos comandos SQL.



**Figura 10:** Ícone que executa uma linha de comando SQL.

Estas duas opções executam apenas o comando em que o cursor se encontra. Para executar todos os comandos digitados na interface, ou seja, um **script**, você pode pressionar a tecla <F5> ou selecionar o ícone apresentado na figura 11.



**Figura 11:** Ícone que executa todas as linhas do script.

Neste momento, como só existe um comando escrito em três linhas, você pode optar por qualquer um dos métodos, desde que se limite à apenas uma execução. Você notará que a janela principal será

### Script

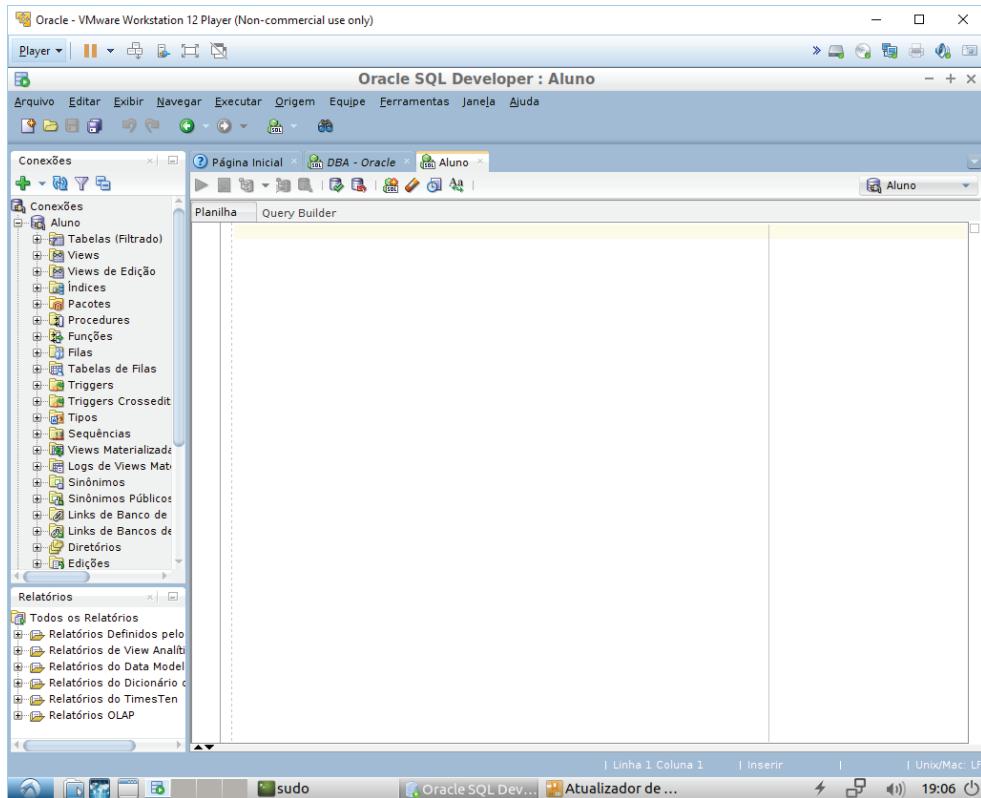
conjunto de comandos sql dispostos em uma lógica sequencial.

dívidida e a Saída do *Script* se tornará visível com a mensagem “User ALUNO criado”.

Após obter sucesso na primeira execução, se você tentar novamente, irá ser notificado de um erro. Isso se dá porque o SGBD teve sucesso na primeira execução criando o usuário e, ao tentar executar o comando pela segunda vez, o SGBD nota que o usuário já existe, sendo impossível criar um novo usuário com o mesmo nome, pois esta ação criaria redundância. Assim o SGBD lhe informa que existe um conflito no nome ou um erro de atribuição. Para dar privilégio para o usuário aluno de criar uma sessão de conexão com o banco de dados, criar tabelas, criar visão de dados digite:

```
GRANT create session, create table, create view, dba TO aluno;
```

Execute somente esta linha de comando com <Ctrl><Enter> e observe se na Saída do *Script* apareceu a mensagem “Grant bem-sucedido”. Se você conseguiu executar ambos os comandos com sucesso, você pode criar uma nova conexão para o usuário aluno. Para criar a conexão do usuário Aluno, selecione Arquivo, Novo, Camada do Banco de Dados e Botão Ok ou se preferir pode utilizar o atalho <Ctrl><N> pressionando as teclas simultaneamente. É possível executar o mesmo comando clicando sobre o + verde que aparece na interface gráfica na janela conexões. Selecione Nova Conexão e uma janela semelhante a apresentada na figura 08 deverá aparecer novamente. No campo nome digite “Aluno”, no campo nome do usuário digite “aluno” e em senha digite “aluno”. Selecione o campo “Salvar Senha” e selecione o botão Testar. Se aparecer no Status: “Sucesso” selecione o botão Conectar. Observe que a nova conexão foi criada, selecione o + que aparece antes da Conexão Aluno e a interface deve ficar semelhante a indicada na figura 12.



**Figura 12:** Conexão Aluno criada.

## ESQUEMA DE BANCO DE DADOS.

O SQL esquema é um grupo de objetos de banco de dados, que podem ser tabelas e índices, relacionados entre si. Na maioria das vezes, o esquema pode ser atribuído a uma aplicação ou usuário. O SGBD pode manter vários esquemas que pertencem a grupos de usuários ou aplicações. Os esquemas são úteis pois agrupam as tabelas por proprietários e aplicam o primeiro nível de segurança, ou seja, o usuário só pode ver as tabelas que lhe pertence. O comando que cria um esquema é:

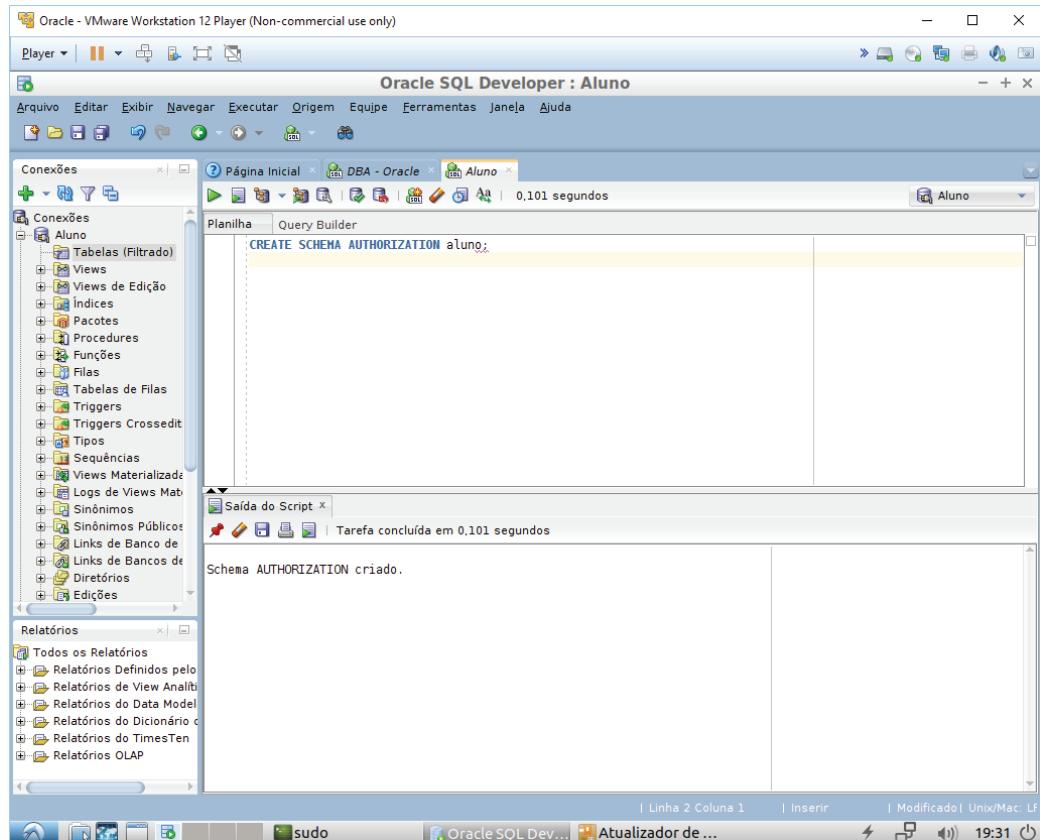
```
CREATE SCHEMA AUTHORIZATION nome_usuario;
```

Então você deve digitar na nova conexão:

```
CREATE SCHEMA AUTHORIZATION aluno;
```

A maioria dos SGBDRs empresariais dá suporte a esse comando, mas eles atribuem o comando automaticamente quando você cria o

usuário pela interface gráfica. Neste momento, você está criando manualmente e vale ressaltar que este comando só funciona se você estiver conectado com o usuário que terá o esquema criado, assim, não tente criar um esquema para o usuário DBA e nem fazê-lo definir o esquema do usuário aluno. Execute o comando como apresentado na figura 13.



**Figura 13:** Criando esquema no usuário aluno.

**Esquema de banco de dados**  
grupo de objetos relacionados entre si.

Após a criação do **esquema de banco de dados**, você poderá definir as estruturas das tabelas que você deseja criar. Para tanto, veja os tipos de dados que você pode utilizar na construção ou parametrização de dados no *Oracle*.

## VARCHAR2 OU VARCHAR.

São dados de caracteres com comprimento variável, podendo atingir o tamanho máximo de 4.000 bytes em uma tabela e 32.767 bytes no PL/SQL. O varchar2 é variável pois somente utiliza espaço ocupado, não fazendo reserva de alocação. É possível utilizar varchar ou varchar2 no SQL. A *Oracle* manteve o tipo varchar para ser compatível com outros

SGBDs, contudo está opção não é recomendada visto que ela pretende utilizar tal sintaxe em suas novas versões à medida que o SQL evolui.

## **CHAR.**

São dados de caracteres com comprimento fixo de até 255 caracteres ou 2.000 bytes. Ao contrário do *varchar2*, o *char* sempre contém o tamanho máximo atribuído a ele, independentemente do dado atribuído, sua alocação sempre será os dados mais os espaços em branco. Seu uso só é vantajoso quando se tem uma cadeia de caracteres obrigatórios como por exemplo as 3 letras da placa de um carro ou o sexo de um usuário.

## **NUMBER**

São dados numéricos com sinal e pode apresentar ponto decimal sendo *NUMBER(n,d)* onde n é o número total de dígitos e d a quantidade de números do total que irão compor o decimal. Assim o n deve ser sempre maior que o d, pois o n é o total de números que compõem o tipo e d define quantos números pertencerão a parte decimal do número. Por exemplo a declaração *NUMBER(6,2)* indica que é possível armazenar 123,56 pois temos o total de 5 números e o símbolo decimal. Também é possível armazenar -43,15 observe que o sinal negativo será computado no tamanho máximo atribuído. O tipo *NUMBER* apresenta alguns subtipos: *DECIMAL*, *DEC*, *DOUBLE PRECISION*, *NUMERIC*, *INTEGER*, *INT*, *SMALLINT*, *FLOAT*. Como *varchar*, os subtipos criados visam tornar compatível e facilitar que usuários de outros SGBD migrem para o *Oracle*; assim, na maioria dos casos esses tipos são *alias* ou apelidos para o tipo *NUMBER*. Assim, se você não está migrando o SGBD, recomenda-se utilizar o *NUMBER* para os dados numéricos.

## **DATA.**

São dados no formato de data e hora. O nome deveria ser *DATATIME* pois a hora está sempre presente independentemente de você utilizá-la ou não. Se você não for utilizar o campo hora, o sistema atribui 00:00 A.M. automaticamente. Permite data entre 1 de janeiro de 4712 AC até 31 de dezembro de 9999 DC. O calendário interno tem em conta as alterações de calendário como a passagem do calendário Juliano para Gregoriano em 1582, onde foram eliminados 10 dias.

## **TIMESTAMP.**

São dados no formato de data, hora com segundos fracionados. É similar ao tipo *data* mas acrescido dos segundos fracionados. Existem muitas variações deste tipo de dados como *WITH TIMEZONE* e *WITH LOCAL TIMEZONE*.

## **BOOLEAN.**

São dados que armazenam 1 para os valores *true* ou 0 para valores *false*.

## **LONG.**

São dados de caracteres de comprimento variável até 2 *gigabytes*. Fornecido para compatibilidade com versões anteriores. No PL/SQL, pode atingir o tamanho máximo de 32.760 *bytes*. Observe que isso é menor do que a largura máxima de uma coluna *LONG* em uma tabela.

## **RAW.**

São dados brutos que podem ser armazenados em uma coluna. O comprimento máximo é de 2.000 *bytes*.

## **LONG RAW.**

São dados brutos parecido com o tipo *RAW* que podem ser armazenados até o tamanho máximo de 2 *GB*. Existem algumas restrições para *LONG RAW* e *LONG*, como uma função não poder retornar o tipo *LONG*, não poder ser indexado, não poder ser cláusula de restrição como *WHERE*.

Por apresentar limitações, a *Oracle* criou o tipo *LOB* para resolver as limitações impostas pelo tipo *LONG*.

## **CLOB.**

São dados do tipo caractere com acentuações no tamanho de 1 até o tamanho máximo de 4 *GB*. O número de colunas *CLOB* por tabela é limitado somente pelo número máximo de colunas por tabela.

## **NCLOB.**

Similar a *CLOB* com suporte a *Unicode*. Conjuntos de caracteres de largura fixa e largura variável são suportados. Armazena dados de conjunto de caracteres nacionais.

## **BLOB.**

Armazenam dados não estruturados como: som, imagem e dados binários. O tamanho máximo é de 4 GB, podendo atingir 128 GB de armazenamento. Os objetos podem ser pensados como *bitstreams* sem semântica. Os objetos possuem suporte transacional completo. Você pode fazer alterações através do SQL com os pacotes *DBMS\_LOB* ou *Oracle Call Interface* (OCI). As manipulações de valor podem ser feitas e revertidas. No entanto, você não pode salvar uma localização em uma sessão e depois usá-la em outra sessão.

## **BFILE.**

Permite o acesso de arquivos binários que são armazenados no sistema de arquivos fora do banco de dados. Um atributo *BFILE* armazena o local do arquivo binário no sistema do servidor. O localizador mantém o nome do diretório e o nome do arquivo. Você pode alterar o nome do arquivo e o caminho sem afetar a tabela. Os arquivos binários *LOB* não participam de transações e não são recuperáveis. Em vez disso, o sistema operacional fornece integridade e o armazenamento do arquivo. O *DBA* deve garantir que o arquivo externo exista e que os processos *Oracle* possuam permissões de leitura no sistema operacional no arquivo. O tipo de dados permite o suporte de somente leitura nos arquivos binários grandes. Você não pode modificar ou replicar esse arquivo. *Oracle* fornece *APIs* para acessar os dados dos arquivos. As interfaces primárias que você usa para acessar os dados do arquivo são os pacotes *DBMS\_LOB* e *Oracle Call Interface* (OCI).

## **RESUMO DOS TIPOS DE DADOS.**

Os tipos de dados mais comuns que você vai utilizar no *Oracle* são apresentados na tabela 05 sendo que estes devem nortear o processo de definição dos atributos ao se criarem as tabelas.

**Tabela 05:** Tipos de Dados Padrão.

TIPO DE DADOS	DESCRIÇÃO
VARCHAR2 (n)	Campo do tipo caractere com tamanho variável.
CHAR (n)	Campo caractere fixo com tamanho máximo de 255 caracteres.
DATE	Campo de data e hora dia, mês, ano, hora, minuto e segundo.
NUMBER (n, d)	Campo numérico onde n é total de dígitos e d as casas decimais.
BLOB	Campo para som, imagem e dados binários.



## leitura indicada

Para mais informações sobre os tipos de dados que você pode utilizar nos atributos, consulte a matéria intitulada Referências da Linguagem SQL para Banco de Dados disponível em:  
[http://docs.oracle.com/cd/B28359\\_01/server.111/b28286/sql\\_elements001.htm#SQLRF0021](http://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements001.htm#SQLRF0021)

e Tipos de Dados do Oracle 8 a Oracle 11g disponível em:  
<https://ss64.com/ora/syntax-datatypes.html>

Quando se define o tipo de dados do atributo, você deve pensar em sua utilização. Será que este dado será utilizado em algum processamento ou apenas em nível de informação? Vai sofrer cálculos? Para ilustrar, imagine uma aplicação imobiliária, um atributo será o número de banheiros de uma casa. Este atributo pode ser *number(2)* ou *varchar2(2)*. Se optar por maior velocidade e menor espaço de armazenamento você vai atribuir *varchar2*, pois não ocorrem operações aritméticas sobre o número de banheiros. Se quiser classificar as casas pelo número de banheiros, você vai optar por *number* para efetuar comparações e criar índices. Assim, vale refletir sobre a utilização do atributo para definir o tipo de maneira mais adequada. Os SGBDRs comerciais dão suporte a muito mais tipos de dados dos que apresentados na tabela 05, no entanto esta aula foi concebida para apresentar os tipos básicos do SQL.

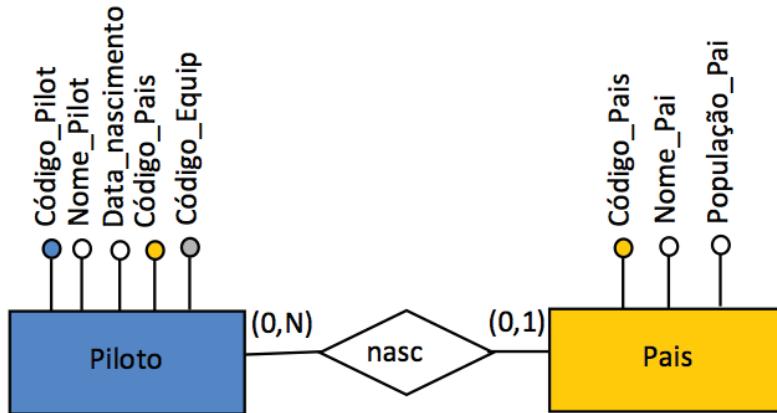
## CRIANDO TABELAS.

Agora você está pronto para criar a primeira tabela no SGBD com a ajuda do comando *CREATE TABLE* apresentado a seguir.

```
CREATE TABLE nome_da_tabela (
    coluna_1      tipo_de_dados [restrição] [, 
    coluna_2      tipo_de_dados [restrição] [, 
    PRIMARY KEY  (coluna_1      [, coluna_2]) ] [, 
    FOREIGN KEY   (coluna_1      [, coluna_2]) REFERENCES 
                  nome_da_tabela] [, 
    CONSTRAINT nome_restricção ]
) ;
```

Para facilitar a leitura do SQL, a maioria dos DBA e programadores utilizam uma linha para definir uma coluna e tabulação para alinhar características e restrições dos atributos. Outra característica é que você pode padronizar seu código mantendo por convenção o nome das tabelas em maiúsculo e atributos em minúsculo.

Para exemplificar, imagine que você deve criar as tabelas representadas no modelo DER apresentado na figura 14.

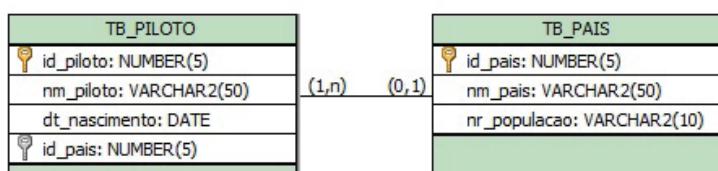


**Figura 14:** Modelo Conceitual das entidades Piloto e País.

A primeira tarefa é converter o Modelo Conceitual em Modelo Lógico. Nesta etapa, você pode aproveitar e remover acentuação, cedilha e adotar um dos padrões estudados na sessão 4 desta aula. O modelo lógico no formato texto ficaria assim:

```
TB_PILOTO(id_piloto, nm_piloto, dt_nascimento, id_pais, id_equip)
TB_PAIS(id_pais, nm_pais, nr_poulacao)
```

O modelo gráfico fica similar ao apresentado na figura 15.



**Figura 15:** Modelo Lógico padronizado com qualificado nome descritivo.

O código SQL para criar a tabela Piloto fica assim:

```
CREATE TABLE TB_PILOTO (
```

```
    id_piloto      NUMBER(5)      PRIMARY KEY,
```

```

nm_piloto      VARCHAR2(50),
dt_nascimento  DATE,
id_pais        NUMBER(5)
);

```

Para executar a instrução, você deve apertar <Ctrl><Enter> simultaneamente ou utilizar o ícone *play*. A instrução cria a tabela TB\_PILOTO com id\_piloto como chave primária, a coluna nm\_piloto com 50 espaços, a coluna dt\_nascimento para armazenar uma data válida e uma coluna que será definida como chave estrangeira mais adiante. Para criar a tabela TB\_PAIS, você pode utilizar um comando similar ao anterior. Você notará que o SQL permite ao desenvolvedor variações de métodos, por isso a seguir será apresentado uma outra forma de indicar a chave primária. Para criar novamente a tabela, é necessário antes apagar a existente. Utilize:

```
DROP TABLE TB_PILOTO;
```

Após executar o comando *DROP TABLE*, você poderá executar novamente o comando *CREATE TABLE* só que neste caso estamos especificando uma mensagem caso ocorra a violação da chave primaria.

```

CREATE TABLE TB_PILOTO (
id_piloto      NUMBER(5) CONSTRAINT tb_piloto_id_piloto_
                _pk PRIMARY KEY,
nm_piloto      VARCHAR2(50),
dt_nascimento  DATE,
id_pais        NUMBER(5)
);

```

Restrições ou *CONSTRAINT* são regras básicas estabelecidas para o preenchimento de uma ou mais colunas da tabela. Existe uma padronização para as *CONSTRAINT*. Observe que após o comando *CONSTRAINT* é atribuída uma mensagem que será utilizada pelo SGBD para dar um *feedback* amigável para o usuário, caso ocorra uma tentativa violar a regra de restrição. No exemplo visto, você está definindo uma regra de restrição de chave primária. O padrão que utilizado foi declarar:

nome\_da\_tabela + nome\_do\_campo + sigla. A sigla utilizada foi pk que é uma abreviação de *PRIMARY KEY* ou chave primária. Você tem toda a liberdade de criar mensagens otimizadas, desde que não ultrapassem 30 caracteres. As restrições são definidas ao final da especificação de cada coluna ou ao final do comando, antes do ponto-e-vírgula. Entre as restrições mais comuns, encontram-se: chaves primárias (pk), chaves únicas (un), chaves estrangeiras (fk), identificadores de campos obrigatórios (nn), condições para valores permitidos para determinado campo (ck).

Você pode verificar se a tabela foi criada com as especificações que solicitou. Para verificar utilize o comando *DESCRIBE* ou *DESC* que vai mostrar ou exibir as estruturas da tabela ou visão que você verá nas próximas aulas. Digite:

```
DESCRIBE TB_PILOTO;
```

Você pode apagar novamente a tabela, para criar novamente de outra forma. Nesta aula, você deve testar as diferentes formas que o SQL permite você criar tabelas, isso vai lhe ajudar a definir o formato que mais lhe agrada bem como facilitar quando for contratado a se ajustar rapidamente à política adotada pela empresa. É comum a empresa optar por um padrão e solicitar que todos que utilizam o SGBD codifiquem de maneira uniforme. Desta forma, *DROP* a tabela *TB\_PILOTO* e crie novamente com o seguinte comando:

```
DROP TABLE TB_PILOTO;
```

```
CREATE TABLE TB_PILOTO (
    id_piloto      NUMBER(5) ,
    nm_piloto      VARCHAR2(50) ,
    dt_nascimento  DATE ,
    id_pais        NUMBER(5) ,
CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_piloto)
) ;
```

A forma apresentada por último é considerada por muitos DBAs como a mais organizada, pois define as restrições no final do comando.

do. Como você pode perceber, espera-se mais de uma restrição em cada tabela e se você quiser tornar o SGBD mais amigável deverá definir cada restrição com *CONSTRAINT*. Se você não definir uma mensagem de erro, o SGBD deve criar uma automática como SYS\_C007082 onde 007082 identifica a restrição. Caso uma violação ocorra, o SGBD irá informar apenas a violação. Assim *DROP* novamente a tabela e crie com o seguinte comando:

```
DROP TABLE TB_PILOTO;

CREATE TABLE TB_PAIS (
    id_pais          NUMBER(5),
    nm_pais          VARCHAR2(50),
    nr_populacao    VARCHAR2(10),
    CONSTRAINT tb_pais_id_pais_pk PRIMARY KEY (id_pais),
    CONSTRAINT tb_pais_nm_pais_nn CHECK (nm_pais IS NOT NULL)
);

CREATE TABLE TB_PILOTO (
    id_piloto        NUMBER(5),
    nm_piloto        VARCHAR2(50),
    dt_nascimento    DATE,
    id_pais          NUMBER(5),
    CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_piloto),
    CONSTRAINT tb_piloto_id_pais_fk FOREIGN KEY (id_pais)
        REFERENCES TB_PAIS (id_pais)
);
```

Observe que a tabela *TB\_PILOTO* apresenta o atributo *id\_pais* que será utilizado para ser chave estrangeira. Para ser chave estrangeira em uma tabela, o atributo deve ser chave primária em outra tabela e o relacionamento entre as entidades deve ser 1:N. Neste caso, o atributo *id\_pais* é chave primária na *TB\_PAIS* e pode ser declarado a *CONSTRAINT*. Observe que *tb\_piloto\_id\_pais\_fk* é o nome da restrição seguido da declaração de qual campo é chave estrangeira pelo comando *FOREIGN KEY* (*id\_pais*). A declaração *REFERENCES* indica qual tabela será consultada para validar a chave estrangeira, no exemplo *TB\_PAIS*.

e dentro dos parênteses indicamos qual a coluna será consultada. Lembre-se que deve ser a chave primária da tabela, ou seja, a chave primária de TB\_PAIS é id\_pais.

Neste tipo de *CONSTRAINT* podem ser criados relacionamentos que utilizem mais de uma coluna como por exemplo a chave primária composta. Para definir o nome da *CONSTRAINT*, continue a utilizar o padrão nome\_tabela+nome\_do\_atributo+tipo\_constraint. Observe algumas regras:

1. caso o tipo de dados da coluna na tabela inicial e na tabela referenciada sejam diferentes, será apresentado um erro;
2. se a tabela referenciada não possuir chave primária, a *foreign key* será estabelecida sobre a chave primária da tabela referenciada;

O uso de chaves estrangeiras garante que não existirão *tuplas* ou linhas órfãs nas tabelas-filhas (tabelas que possuem dados que devem estar cadastrados previamente em outra tabela, denominada tabela mãe).

Se você tentar apagar a TB\_PAIS com o comando *DROP TABLE TB\_PAIS*, observará uma mensagem de erro. O SGBD agora sabe que a tabela TB\_PAIS é consultada e tem vínculo devido a chave estrangeira com a TB\_PILOTO, de forma que, para apagar a TB\_PAIS, você deve primeiro apagar a TB\_PILOTO e depois a TB\_PAIS. Experimente:

```
DROP TABLE TB_PILOTO;  
DROP TABLE TB_PAIS;
```

Agora você vai experimentar definir um campo como único. Para isso, você pode utilizar o script que estamos trabalhando. Você vai definir uma ou mais colunas que não podem ter valor repetido em mais de uma linha da tabela. Por exemplo, não existem dois países com o mesmo nome, mas este campo não é a chave primária da TB\_PAIS. Então experimente:

```
CREATE TABLE TB_PAIS (  
    id_pais          NUMBER(5),  
    nm_pais          VARCHAR2(50) UNIQUE,  
    nr_populacao    VARCHAR2(10),  
    CONSTRAINT tb_pais_id_pais_pk PRIMARY KEY (id_pais),  
    CONSTRAINT tb_pais_nm_pais_nn CHECK (nm_pais IS NOT  
NULL)  
);
```

Ou pode declarar a *CONSTRAINT*

```
CREATE TABLE TB_PAIS (
    id_pais          NUMBER(5),
    nm_pais          VARCHAR2(50),
    nr_populacao    VARCHAR2(10),
    CONSTRAINT tb_pais_id_pais_pk PRIMARY KEY (id_pais),
    CONSTRAINT tb_pais_nm_pais_nn CHECK (nm_pais IS NOT
NULL),
    CONSTRAINT tb_pais_nm_pais_un UNIQUE (nm_pais)
) ;
```

Outra definição que você pode atribuir a um campo é definir um valor padrão que, se o usuário não informar ao inserir um registro, o SGBD insere automaticamente. Para isso, utilizamos o comando *DEFAULT*. Para exemplificar, se um piloto for cadastrado sem o campo dt\_nascimento, o comando a seguir irá fazer com que o SGBD pegue a data e hora atual do servidor e preencha o campo.

```
CREATE TABLE TB_PILOTO (
    id_piloto        NUMBER(5),
    nm_piloto        VARCHAR2(50),
    dt_nascimento   DATE DEFAULT SYSDATE,
    id_pais          NUMBER(5),
    CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_pilo-
to),
    CONSTRAINT tb_piloto_id_pais_fk FOREIGN KEY (id_pais)
        REFERENCES TB_PAIS (id_pais)
) ;
```

Outra restrição que você pode definir é o sexo do piloto. Imaginando inicialmente que serão classificados em dois gêneros: M para masculino e F para feminino. Você pode apagar a tabela TB\_PILOTO e criá-la com o comando:

```
CREATE TABLE TB_PILOTO (
    id_piloto        NUMBER(5),
    nm_piloto        VARCHAR2(50),
    dt_nascimento   DATE DEFAULT SYSDATE,
    id_pais          NUMBER(5),
```

```

ie_sexo      CHAR(1),
CONSTRAINT tb_piloto_ie_sexo_ck    CHECK (ie_sexo IN
('M','F')) ,
CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_pilo-
to) ,
CONSTRAINT tb_piloto_id_pais_fk    FOREIGN KEY (id_pais)
REFERENCES TB_PAIS (id_pais)
);

```

A *CONSTRAINT CHECK* define um conjunto de valores permitidos ou uma condição para a inserção de dados em uma determinada coluna. Neste caso, o comando *IN* cria uma lista onde qualquer elemento da lista pode ser inserido. Você pode ainda determinar uma expressão ou regra de validação em um *CHECK*. A sintaxe da *CONSTRAINT CHECK* é expressa:

**CONSTRAINT nome\_da\_constraint CHECK (nome\_da\_coluna condição)**

Onde:

*nome\_da\_constraint* é a mensagem que será exibida caso a restrição seja violada;

*nome\_da\_coluna* é a coluna que será validade e testada;  
*condição* é a expressão SQL que deve ser validada.

Por exemplo:

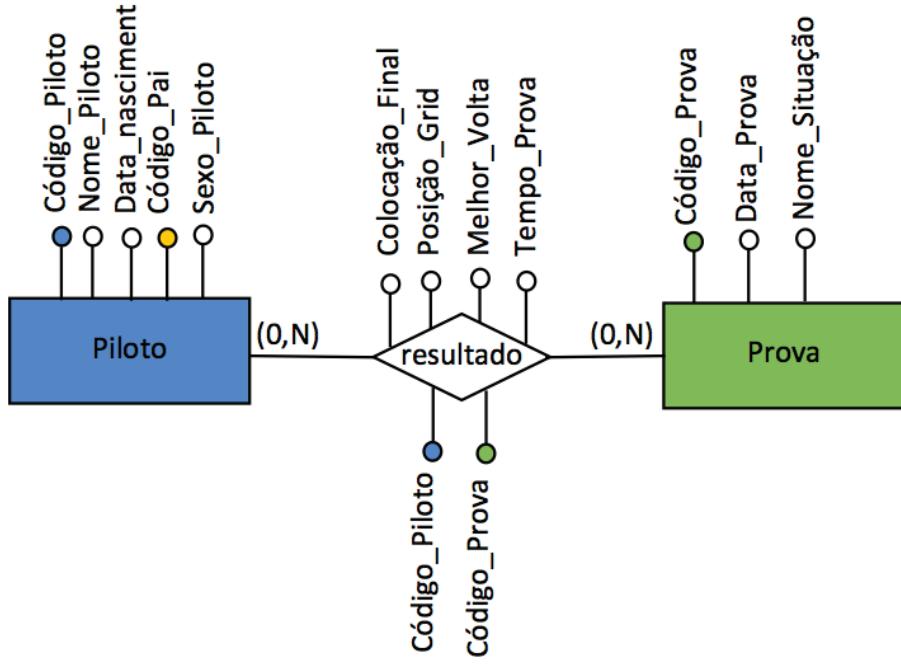
```

CREATE TABLE TB_PILOTO (
id_piloto      NUMBER(5),
nm_piloto      VARCHAR2(50),
dt_nascimento  DATE DEFAULT SYSDATE,
id_pais        NUMBER(5),
ie_sexo        CHAR(1),
CONSTRAINT tb_piloto_ie_sexo_ck    CHECK (ie_sexo IN
('M','F')) ,
CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_pilo-
to) ,
CONSTRAINT tb_piloto_id_pais_fk    FOREIGN KEY (id_pais)
REFERENCES TB_PAIS (id_pais),
CONSTRAINT tb_piloto_nm_piloto_ck CHECK (nm_piloto =
UPPER(nm_piloto))
);

```

Neste caso, a *CONSTRAINT* *tb\_piloto\_nm\_piloto\_ck* vai validar e só irá aceitar inserir dados se os nomes dos pilotos forem cadastrados com todos os caracteres maiúsculos. Caso o usuário tente inserir ou alterar o nome de um piloto com caracteres minúsculos, o SGBD irá apontar a violação da regra e a transação será recusada.

Agora imagine que você deseja acrescentar as seguintes tabelas:



Temos o SQL:

```

CREATE TABLE TB_PROVA (
    id_prova      NUMBER(5),
    dt_prova      DATE,
    st_prova      VARCHAR2(20),
    CONSTRAINT tb_prova_id_prova_pk PRIMARY KEY (id_prova)
);
    
```

O comando é simples e fácil de implementar. O que **é importante** destacar neste exemplo é a próxima tabela que será uma tabela de um relacionamento N:N. Neste caso, a chave primária será composta por duas colunas, sendo que estas colunas são chave estrangeiras.

```

CREATE TABLE TB_RESULTADO (
    id_piloto          NUMBER(5) ,
    id_prova           NUMBER(5) ,
    nr_colocacao_final NUMBER(2) ,
    nr_posicao_grid    NUMBER(2) ,
    hr_tempo_prova    TIMESTAMP ,
    nr_melhor_volta   TIMESTAMP ,
    CONSTRAINT tb_resultado_pk      PRIMARY KEY (id_piloto,id_
prova),
    CONSTRAINT tb_resultado_id_piloto_fk FOREIGN KEY (id_piloto)
        REFERENCES TB_PILOTO (id_piloto),
    CONSTRAINT tb_resultado_id_prova_fk FOREIGN KEY (id_prova)
        REFERENCES TB_PROVA (id_prova)
);

```

Observe como a chave primária foi declarada com duas colunas, esta é uma chave primária composta. As chaves primárias podem ter inúmeras colunas, sendo definidas no momento do projeto.

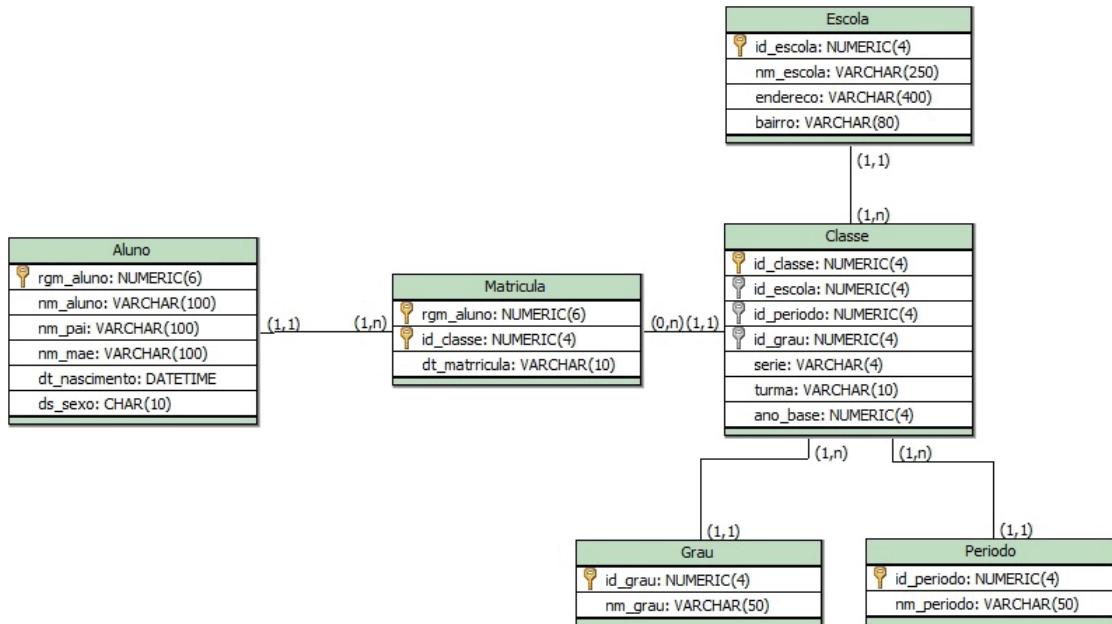


**síntese**

Nesta quinta aula, você conheceu as principais etapas na normalização de um banco de dados relacional e pode compreender quais os custos e benefícios de se normalizar um banco de dados. Você pode conhecer dois modelos de padronização e como as ferramentas case podem lhe ajudar. Por fim, você iniciou o SQL, criando sua conexão, usuário, aprendendo como representar os tipos de dados no SQL e criando suas primeiras tabelas.

## Atividades

01. Com base no esquema diagramático de Engenharia da Informação defina o script SQL



---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# AULA 6

## INTRODUÇÃO AO SQL

### NESTA AULA

- » Alterando tabelas existentes.
- » INSERT e conceitos iniciais da SELECT.
- » Consultas utilizando SELECT.
- » Expressões aritméticas e os operadores BETWEEN, IN, LIKE e IS NULL.
- » Operadores Especiais
- » UPDATE e DELETE..

### ■ METAS DE COMPREENSÃO

- » Conhecer como se altera as tabelas existentes.
- » Utilizar os comandos para criar pontos de referência e desfazer alterações.
- » Aprender como inserir, alterar, excluir e recuperar dados no SGBD.

### ■ APRESENTAÇÃO

Nesta aula você vai aprender a alterar tabelas que já existem no banco de dados. Você entenderá como inserir, alterar e excluir registros utilizando o SQL. Conhecerá os comandos COMMIT e ROLLBACK que vão lhe ajudar a desfazer algumas operações que podem ocorrer. Também você iniciará o comando SELECT que retorna dados que foram armazenados em tabelas. Por fim, você vai ver como testar as restrições que você criou e aprender a utilizar os operadores do SQL.

# ■ ALTERANDO TABELAS EXISTENTES

**ALTER TABLE**  
comando SQL que permite alterações na estrutura de uma tabela.

**ADD**  
adiciona colunas ou restrições em uma tabela.

**MODIFY**  
altera as características de uma tabela.

**DROP**  
exclui colunas ou restrições em uma tabela.

Nesta aula, você vai aprender como alterar estruturas de tabelas mudando características de atributos e adicionando colunas. Todas as alterações na estrutura de uma tabela são feitas utilizando o comando **ALTER TABLE**, seguido por uma palavra que informa o tipo de alteração que se deseja. Existem três opções diferentes para alterar. A opção **ADD** adiciona uma coluna ou restrição, **MODIFY** altera as características de uma coluna e **DROP** exclui uma coluna ou restrição da tabela. Se a tabela estiver vazia todas as operações serão executadas sem problemas, mas se a tabela conter registro, o SGBD vai verificar se a ação é possível de ocorrer sem perda de dados, se não for, uma mensagem negando a operação será exibida. Assim, operações como **DROP** em uma coluna com registros será sempre negada. A sintaxe básica para se alterar uma tabela inserindo é:

```
ALTER TABLE nome_da_tabela  
ADD [nome_da_coluna tipo_de_dados] | [restrição]
```

Para exemplificar, imagine que você tenha criado a tabela TB\_PROVA com o seguinte código:

```
CREATE TABLE TB_PROVA (  
    id_prova NUMBER(5),  
    dt_prova DATE,  
    st_prova VARCHAR2(20),  
    CONSTRAINT tb_prova_id_prova_pk PRIMARY KEY (id_prova)  
) ;
```

Em algum momento durante a análise, você percebe que será melhor ter o atributo hr\_prova separado de dt\_prova para gerar relatórios. Para alterar a tabela e acrescentar uma nova coluna utilize:

```
ALTER TABLE TB_PROVA  
ADD hr_prova DATE;
```

Observe que para modificar um campo que não estava como obrigatório para obrigatório, basta utilizar o código:

```
ALTER TABLE TB_PROVA  
MODIFY dt_prova NOT NULL;
```

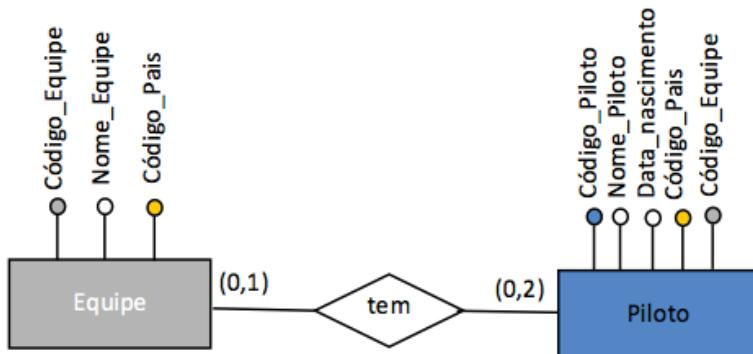
Se você quiser modificar o tipo de um atributo, utilize:

```
ALTER TABLE TB_PROVA  
MODIFY hr_prova TIMESTAMP;
```

A modificação do tamanho de um tipo de dados se dá com o seguinte código:

```
ALTER TABLE TB_PROVA  
MODIFY st_prova VARCHAR2(10);
```

Note que você pode aumentar o tamanho mesmo existindo registro na tabela; contudo, a redução só é possível se todos os registros não utilizam o tamanho especificado. No exemplo, o campo st\_prova tinha a possibilidade de ter até 20 caracteres, a redução para 10 só será efetivada após o SGBD verificar se nenhum dado será perdido com a solicitação. Uma utilização muito comum, é utilizar o comando *ALTER TABLE* para adicionar as restrições ao se criar uma tabela. Isso se dá para facilitar encontrar erros ao se criar as tabelas. Para ilustrar, imagine que você agora vai acrescentar as tabelas apresentadas na figura 01 na base de dados.



**Figura 01:** DER F1 acrescentando TB\_EQUIPE.

Assim poderíamos criar a TB\_EQUIPE como exemplificado a seguir.

```

CREATE TABLE TB_EQUIPE (
    id_equipe      NUMBER(5),
    nm_equipe      VARCHAR2(50),
    id_pais        NUMBER(5)
);

```

Observe que no exemplo você não informou qual é a chave primária, qual é a chave estrangeira, se terá campos obrigatórios ou qualquer outra restrição. Você vai utilizar o comando *ALTER TABLE* para definir cada uma dessas restrições. Assim, se você errar ao declarar uma delas, será mais fácil encontrar o erro e corrigi-lo. Segue como fica o restante do código:

```

ALTER TABLE TB_EQUIPE
    ADD CONSTRAINT tb_equipe_id_equipe_pk
        PRIMARY KEY (id_equipe);

ALTER TABLE TB_EQUIPE
    ADD CONSTRAINT tb_equipe_id_pais_fk
        FOREIGN KEY (id_pais)
        REFERENCES TB_PAIS (id_pais);

ALTER TABLE TB_EQUIPE
    ADD CONSTRAINT tb_equipe_nm_equipe_nn
        CHECK (nm_equipe IS NOT NULL);

```

Na aula anterior, você já tinha criado a TB\_PILOTO com o código:

```

CREATE TABLE TB_PILOTO (
    id_piloto      NUMBER(5),
    nm_piloto      VARCHAR2(50),
    dt_nascimento  DATE DEFAULT SYSDATE,
    id_pais        NUMBER(5),
    ie_sexo        CHAR(1),
    CONSTRAINT tb_piloto_ie_sexo_ck          CHECK (ie_sexo IN
('M','F')),
    CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_piloto),
    CONSTRAINT tb_piloto_id_pais_fk   FOREIGN KEY (id_pais)
        REFERENCES TB_PAIS (id_pais),
    CONSTRAINT tb_piloto_nm_piloto_ck  CHECK (nm_piloto =
UPPER(nm_piloto))
);

```

Observe que neste caso ficou faltando uma coluna *id\_equipe*, assim altere com o código:

```
ALTER TABLE TB_PILOTO
```

```

ADD id_equipe NUMBER(5);

ALTER TABLE TB_PILOTO
ADD CONSTRAINT tb_piloto_id_equipe_fk
FOREIGN KEY (id_equipe)
REFERENCES TB_EQUIPE (id_equipe);

```

Também é possível utilizar o comando *ALTER TABLE* para remover uma coluna ou restrição. A sintaxe seria a seguinte:

```

ALTER TABLE nome_da_tabela
DROP [PRIMARY KEY|COLUMN nome_da_coluna|CONSTRAINT nome_
restrição];

```

Observe que para remover uma restrição, você deve informar o nome da restrição. Por isso, uma boa prática é sempre criar restrições com nome e não deixar que o SGBD crie de forma automática. Se acontecer de você deixar o SGBDR *Oracle* criar e precisar consultar o nome atribuído, utilize o comando *SELECT* apresentado a seguir.

```

SELECT * FROM USER_CONSTRAINTS
WHERE table_name='TB_PROVA';

```

Onde aparece o TB\_EQUIPE você poderá substituir pelo nome de qualquer tabela que desejar consultar. Outro comando semelhante é:

```

SELECT constraint_name, constraint_type
FROM USER_CONSTRAINTS
WHERE table_name='TB_PROVA';

```

Algumas informações sobre as restrições que foram criadas para uma tabela são armazenadas na tabela de controle e atualizada automaticamente. Esta tabela é a *USER\_CONSTRAINTS*. O comando *SELECT* faz uma busca dos dados cadastrados lhe exibindo o resultado. Agora que você sabe o nome de suas restrições você pode experimentar apagar uma coluna com o comando:

```

ALTER TABLE TB_PROVA
DROP COLUMN hr_prova;

```

O comando vai apagar a coluna hr\_prova da tabela. Para validar a alteração, utilize o comando:

```
DESC TB_PROVA;
```

Para apagar uma restrição utilize:

```
ALTER TABLE TB_PROVA
    DROP CONSTRAINT SYS_C007125;
```

Observe que, como não definimos o nome da *CONSTRAINT* quando solicitamos que o SGBD alterasse a TB\_PROVA tornando o campo dt\_prova obrigatório, o SGBD atribuiu o nome de forma automática. No seu sistema pode ter criado outro nome, assim use a instrução *SELECT* anterior para verificar. Como você percebeu, **não é bom deixar nomes estranhos, assim utilize o comando para corrigir:**

```
ALTER TABLE TB_PROVA
    ADD CONSTRAINT tb_prova_dt_prova_nn
        CHECK (dt_prova IS NOT NULL);
```

Para finalizar o comando *ALTER TABLE*, vale ressaltar que em alguns momentos você pode precisar desabilitar algumas restrições criadas no banco de dados. São momentos em que você vai ter uma grande carga de processamento, normalmente de balanço ou exportação de dados para uma base de estatística, que não será necessário usar normalização. Se você tiver certeza que não ocorrerá modificação na base de dados, você pode acelerar o processo desabilitando algumas restrições ou todas. O comando a seguir desabilita uma restrição:

```
ALTER TABLE nome_da_tabela
    DISABLE nome_da_restricao;
```

Por exemplo, para desabilitar a chave estrangeira da TB\_EQUIPE digite:

```
ALTER TABLE TB_EQUIPE
    DISABLE CONSTRAINT tb_prova_id_pais_fk;
```

Para habilitar a chave estrangeira digite:

```
ALTER TABLE TB_EQUIPE
```

```
ENABLE CONSTRAINT tb_prova_id_pais_fk;
```

Se você desativar uma restrição de chave única ou primária que esteja usando um índice exclusivo, a *Oracle* descartará o índice exclusivo. Quando você habilitar a chave primária, o SGBD vai criar o índice automaticamente. Com o comando *ALTER TABLE*, você consegue mudar a tabela do modo gravação para consulta.

```
ALTER TABLE TB_EQUIPE READ ONLY;
```

Este comando não muda a tabela para somente leitura, impedindo que alterações com comandos DML e DDL ocorram. Para voltar a tabela para o modo gravação, utilize o comando apresentado a seguir.

```
ALTER TABLE TB_EQUIPE READ WRITE;
```

A tabela volta ao modo gravação, permitindo que se insira, altere, apague sua estrutura e dados.

## ■ INSERT E CONCEITOS INICIAIS DA SELECT

Para inserir uma linha em uma tabela SQL, é necessário a utilização do comando *INSERT*. A sintaxe básica do *INSERT* é:

```
INSERT INTO nome_da_tabela [(nome_da_coluna [, nome_da_coluna])]  
VALUES (valor_da_coluna [, valor_da_coluna]);
```

Para exemplificar, veja como você pode inserir um país na TB\_PAIS.

```
INSERT INTO TB_PAIS VALUES (1, 'Brasil', '207.7 mi');
```

No exemplo, foi informado o nome\_da\_tabela como TB\_PAIS. Observe que não foram atribuídas as informações de quais colunas e em qual ordem seriam enviados os dados no **VALUES**. Assim, o SGBD assume que serão enviados todos os dados na ordem que foram infor-

### VALUES

apresenta os valores.

mados quando se criou a tabela. A instrução *VALUES* passa a informar os valores, sendo a ordem *id\_pais*, *nm\_pais*, *nr\_populacao*. O *id\_pais* é a chave primária, isso quer dizer que ele não pode ser nulo ou ser um valor repetido na tabela. Ele é do tipo *NUMBER* o que dispensa o uso de aspas simples. Todos os dados do tipo *VARCHAR2* devem ser informados com as aspas simples e não podem ultrapassar o limite ou tamanho que foi definido na estrutura da tabela. Veja uma variação que você pode utilizar.

```
INSERT INTO TB_PAIS VALUES (2, 'Alemanha', NULL);
```

Neste caso, foi escolhido que seriam enviados todos os dados na ordem em que se criou a tabela, mas no *VALUES* o usuário optou por deixar o valor do atributo *nr\_populacao* nulo utilizando a instrução **NULL** para esse fim. Outra forma de inserir seria:

```
INSERT INTO TB_PAIS (id_pais, nm_pais) VALUES (3, 'Finlândia');
```

Você pode mudar a ordem em que os dados serão informados, como segue no próximo comando.

```
INSERT INTO TB_PAIS (nm_pais, id_pais) VALUES ('França', 4);
```

Pode conter todos os parâmetros em uma outra ordem.

```
INSERT INTO TB_PAIS (nr_populacao, id_pais, nm_pais)
VALUES ('5.731 mi', 5, 'Dinamarca');
```

Pode também informar a ordem e deixar um valor nulo.

```
INSERT INTO TB_PAIS (nr_populacao, id_pais, nm_pais)
VALUES (NULL, 6, 'Bélgica');
```

Observe que o valor nulo só será aceito se no momento em que você definiu as *CONSTRAINT*, você não criou uma restrição de campo obrigatório ou *NOT NULL*. Para inserir múltiplas linhas, utilize o comando **INSERT ALL**.

```
INSERT ALL
```

```
INTO TB_PAIS (id_pais, nm_pais) VALUES (7, 'Espanha')
INTO TB_PAIS (id_pais, nm_pais) VALUES (8, 'Reino Unido')
INTO TB_PAIS (id_pais, nm_pais) VALUES (9, 'Austrália')
SELECT * FROM DUAL;
```

O comando *INSERT ALL* é usado para adicionar várias linhas com uma única instrução *INSERT*. As linhas podem ser inseridas em uma tabela ou em várias tabelas usando apenas um comando SQL. No exemplo inserimos três linhas ou tuplas na TB\_PAIS. Observe que no final do comando *INSERT ALL* existe uma *SELECT* da tabela *DUAL*. Esta tabela apresenta uma única coluna chamada *DUMMY* e um único registro. Ela foi criada no usuário *SYS* e sua função é auxiliar em operações como a do comando *INSERT ALL*. Mais adiante você verá outras operações que a tabela *DUAL* pode auxiliar. Outra forma de utilizar o *INSERT ALL* é utilizar a ordem padrão da tabela.

```
INSERT ALL
INTO TB_PAIS VALUES (10, 'Países Baixos', NULL)
INTO TB_PAIS VALUES (11, 'Suécia', NULL)
INTO TB_PAIS VALUES (12, 'Canadá', NULL)
INTO TB_PAIS VALUES (13, 'Itália', NULL)
INTO TB_PAIS VALUES (14, 'Rússia', NULL)
SELECT * FROM DUAL;
```

## ■ CONSULTAS UTILIZANDO SELECT

Para verificar como os dados, ou o conteúdo que está na tabela, você deve utilizar o comando *SELECT*. Este é principal comando da SQL e serão necessárias diversas interações com o banco de dados para absorver todas as possibilidades e combinações implementadas nesta instrução. Para iniciar, observe uma descrição simplificada de sua sintaxe.

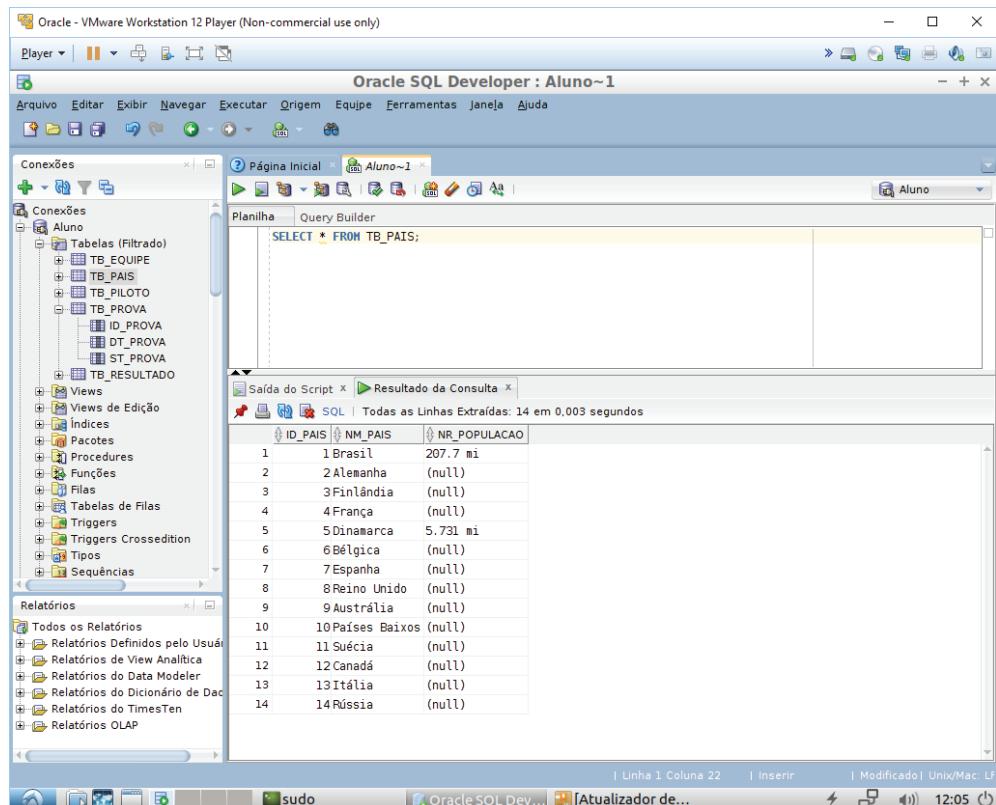
```
SELECT lista_de_colunas FROM nome_da_tabela;
```

Esta é uma sintaxe bem simplificada que introduz a *SELECT* onde a *lista\_de\_colunas* representa um ou mais atributos que devem ser exibidos. Se você optar por solicitar mais de uma coluna, você deve separar as colunas utilizando uma vírgula. Também é permitido utilizar o caracte-

ter o asterisco como coringa, ou seja, todas as colunas da tabela na ordem de criação participaram da consulta. Por exemplo, se você deseja ver todos os países que cadastrou na TB\_PAIS pode utilizar:

```
SELECT * FROM TB_PAIS;
```

O resultado é apresentado na figura 02 do comando *SELECT \* FROM TB\_PAIS*.



**Figura 02:** Resultado da *SELECT \* FROM TB\_PAIS*.

Embora o comando *SELECT* possa ser agrupado em uma única linha, em outras consultas as sequências vão ficar mais complexas e por isso serão apresentadas em linhas separadas. Utilizar uma convenção para escrever e ler SQL torna mais fácil a manutenção, rastreamento lógico, correções e o reuso de código SQL.

Ao executar uma consulta a uma tabela o SGBD, retorna um conjunto de linhas que apresentam as mesmas características que uma tabela. A *SELECT* é orientada para atuar sobre o conjunto de linhas podendo atuar a uma ou mais colunas e zero ou mais linhas de uma tabela. Para solicitar apenas os nomes dos países cadastrados na TB\_PAIS escreva:

```
SELECT nm_pais FROM TB_PAIS;
```

Quando mostramos o resultado de uma pesquisa, normalmente é retornado o nome das colunas selecionadas como cabeçalho. Observe na figura 03 como é apresentado o nome do campo NM\_PAIS.

The screenshot shows the Oracle SQL Developer interface. In the center, there's a 'Query Builder' window with the following SQL code:

```
SELECT nm_pais FROM TB_PAIS;
```

Below the query, the results are displayed in a table with one column labeled 'NM\_PAIS'. The data consists of 14 rows of country names:

NM_PAIS
1 Espanha
2 Brasil
3 Alemanha
4 Finlândia
5 França
6 Dinamarca
7 Bélgica
8 Reino Unido
9 Austrália
10 Países Baixos
11 Suécia
12 Canadá
13 Itália
14 Rússia

**Figura 03:** Nome do campo NM\_PAIS.

Muitas vezes não é aceitável exibir o nome do atributo ou o retorno como título da coluna. Para alterar o nome da coluna, é possível criar um apelido para a coluna ou **ALIAS**. O **ALIAS** irá substituir o nome da coluna, protegendo a estrutura do SGBD e facilitando a exibição em relatórios gerados pelos usuários. A sintaxe do **ALIAS** é bem simples, você poderá optar por utilizar o comando **AS** ou não. Se você quiser substituir o nome da coluna por apenas uma palavra simples, basta digitar após o nome da coluna; mas, caso queira utilizar uma sentença ou caracteres especiais, deverá utilizar aspas-duplas para indicar que o nome deve ser substituído pela sentença. Veja a sintaxe do comando **ALIAS**:

```
SELECT nome_da_coluna [AS] apelido FROM nome_da_tabela;
```

### AS

apelido que pode ser atribuído a uma coluna ou ao nome de uma tabela.

Onde:

AS é opcional e serve para indicar que depois virá o apelido da coluna;

apelido pode ser uma única palavra e neste caso não será necessário utilizar aspas-duplas ou poderá ser uma sentença entre aspas-duplas.

Para exemplificar, veja todas as variações permitidas

```
SELECT nm_pais nome FROM TB_PAIS;
SELECT nm_pais AS nome FROM TB_PAIS;
SELECT nm_pais "Nome" FROM TB_PAIS;
SELECT nm_pais AS "Nome" FROM TB_PAIS;
SELECT nm_pais "Nome do País" FROM TB_PAIS;
SELECT nm_pais AS "Nome do País" FROM TB_PAIS;
```

Todas as variações apresentadas acima são aceitáveis e você deve estar preparado para utilizar a definida como padrão para o SGBD. Além do uso do ALIAS em uma *SELECT*, é possível utilizar os operadores aritméticos em um atributo, quer seja numérico ou data. Para exemplificar, você vai inserir na TB\_PILOTO o piloto Nico Hülkenberg. Para inserir o atributo dt\_nascimento será necessário utilizar a instrução **TO\_DATE** que converte uma sequência de caracteres de texto para o formato data. Digite a instrução:

**TO\_DATE**  
instrução que converte string em data.

```
INSERT INTO TB_EQUIPE VALUES (1,'Renault',4);

INSERT INTO TB_PILOTO VALUES (
1, --id_piloto chave primária
'NICO HÜLKEMBERG', --nm_piloto deve escrever em maiúsculas
TO_DATE('19/08/1987', 'dd/mm/yyyy'), --dt_nascimento
3, --id_pais pais em que nasceu
'M', --ie_sexo pode ser M ou F
1 --id_equipe chave estrangeira
);
```

Observe a única instrução nova é o *TO\_DATE* que é responsável por converter a *string* para o formato *date*. A sintaxe do *TO\_DATE* é:

```
TO_DATE( string [,mascara]);
```

Onde:

*string* é o texto que você quer converter para o formato *DATE*; máscara é um campo opcional no qual você informa o que o sistema deve encontrar na *string*.

A máscara pode apresentar uma combinação de valores como apresentado na tabela 01.

**Tabela 01:** Elementos que podem ser utilizados na máscara.

PARÂMETRO	EXPLICAÇÃO
YEAR	Enunciado do Ano.
YYYY	Ano de 4 dígitos.
YY	3, 2 ou último dígito (s) do ano.
Y	
IYY	3, 2 ou último dígito (s) do ano no padrão ISO.
IY	
I	
IYYY	Ano de 4 dígitos com base no padrão ISSO.
RRRR	Aceita um ano de 2 dígitos e retorna um ano de 4 dígitos. Um valor entre 0-49 retornará um ano 20xx. Um valor entre 50-99 retornará um ano 19xx.
Q	Trimestre do ano (1, 2, 3, 4; JAN-MAR = 1).
MM	Meses de 01-12 sendo JAN = 01.
MON	Nome abreviado do mês.
MONTH	Nome do mês, preenchido com espaços em branco com um comprimento de 9 caracteres.
RM	Numeral Romano (I-XII; JAN = I).
WW	Semana do ano (1-53), onde a primeira semana começa no primeiro dia do ano e continua até o sétimo dia do ano.

**Tabela 01:** Continuação.

PARÂMETRO	EXPLICAÇÃO
W	Semana do mês (1-5), onde a primeira semana começa no primeiro dia do mês e termina no sétimo.
IW	Semana do ano (1-52 ou 1-53) com base no padrão ISO.
D	Dia da semana (1-7).
DAY	Nome do dia.
DD	Dia do mês (1-31).
DDD	Dia do ano (1-366).
DY	Nome abreviado do dia.

PARÂMETRO	EXPLICAÇÃO
J	Dia juliano, o número de dias desde 01 de janeiro de 4712 AC.
HH	Hora do dia (1-12).
HH12	Hora do dia (1-12).
HH24	Hora do dia (0-23).
MI	Minuto (0-59).
SS	Segundo (0-59).
SSSSS	Segundos após a meia-noite (0-86399).
AM, A.M., PM ou P.M.	Indicador de meridiano.
AD ou A.D.	Indicador Ano <i>Domini</i> ou Era Comum ou E.C.
BC ou B.C.	Indicador de <i>Before Christ</i> , ou seja, Antes de Cristo ou A.C.
TZD	Informações sobre a economia diurna. Por exemplo, 'PST'.
TZH	Horário do fuso horário.
TZM	Minuto do fuso horário.
TZR	Região do fuso horário.

Após ter inserido o piloto, você pode experimentar fazer cálculos com operadores aritméticos na *SELECT*. É possível utilizar operadores aritméticos nos atributos ou em uma expressão condicional. Expressões Aritméticas podem conter nome de colunas, valores numéricos constantes e operadores aritméticos. Os operadores são apresentados na tabela 02.

**Tabela 02:** Operadores Aritméticos.

OPERADORES	DESCRIÇÃO
+	Adição.
-	Subtração.
*	Multiplicação.
/	Divisão.

É importante não confundir o símbolo de multiplicação com o coringa. Algumas implementações do SQL utilizam o caractere asterisco para ambas as funções e consegue diferenciar seu uso pela lógica dos parâmetros passados. Ao utilizar os operadores aritméticos, todas as regras de precedências são válidas e caso queira modificar a ordem, utilize os parênteses. Apenas para relembrar, se você não modificar a ordem, as operações serão realizadas como apresentado na tabela 03.

**Tabela 03:** Ordem de execução das operações aritméticas.

ORDEM	OPERAÇÕES
1º	Entre parênteses .
2º	Potencialização.
3º	Multiplicações e divisões.
4º	Somas e subtrações.

Para experimentar, teste o seguinte comando:

```
SELECT dt_nascimento+7 AS "Uma semana após nascer" FROM TB_PILOTO;
```

Você pode experimentar utilizar outra operação, por exemplo, para atual idade do piloto.

```
SELECT (sysdate-dt_nascimento)/365.2425 AS "Idade" FROM TB_PILOTO;
```

Neste exemplo, o SGBD irá pegar a data atual do servidor e subtrair a data de nascimento apresentando o resultado em dias. Após a operação que estava entre parênteses, o SGBD irá calcular a idade em anos, por realizar a divisão dos dias vividos por 365.2425 para obter a idade em anos. Além das operações aritméticas é possível concatenar colunas com outras colunas ou com strings utilizando ||. A sintaxe é:

```
SELECT coluna||[coluna/string] AS alias FROM nome_da_tabela;
```

Para exemplificar, experimente:

```
SELECT nm_piloto || ' nasceu em ' || dt_nascimento  
      AS "Nascimento dos pilotos da F1"  
FROM TB_PILOTO;
```

# ■ EXPRESSÕES ARITMÉTICAS E OS OPERADORES BETWEEN, IN, LIKE E IS NULL

Agora que você tem uma noção básica do funcionamento do *SELECT*, você pode conhecer a sintaxe da instrução. O comando *SELECT* apresenta a seguinte sintaxe:

```
SELECT [distinct] col1 [alias], coln
FROM tab1 [alias], tabn [alias]
WHERE condição
GROUP BY colunas
HAVING condição
ORDER BY expressão ou chave [desc];
```

Não se preocupe, você vai estudar e ver cada um dos comandos apresentados na sintaxe do *SELECT*. A Cláusula *WHERE* indica qual condição de execução o *SELECT* deve retornar. Ela funciona com os operadores apresentados na tabela 04.

**Tabela 04:** Operadores para cláusula *WHERE*.

OPERADOR	DESCRIÇÃO
=	Igual.
<>	Diferente.
>	Maior.
<	Menor.
>=	Maior ou igual.
<=	Menor ou igual.

Para exemplificar, você pode solicitar que sejam exibidos todos os países que tenha o nome Brasil cadastrado na tabela.

```
SELECT nm_pais AS "Nome do País", id_pais AS "Código do País"
FROM TB_PAIS
WHERE nm_pais = 'Brasil';
```

Observe que nas condições, os dados alfanuméricos e datas presen-

tes na cláusula *WHERE* devem estar entre aspas simples. O SQL tem três operadores lógicos que são apresentados na tabela 05.



### pense nisso

No mundo real, a busca de dados normalmente vai envolver diversas condições. Por exemplo, você pode querer saber quais pilotos nasceram em um determinado país e tem mais do que 25 anos. Para isso será necessário utilizar operadores lógicos. Os operadores *AND* e *OR* devem ser usados para fazer composições de expressões lógicas. O predicado *AND* esperará que ambas as condições sejam verdadeiras enquanto que o predicado *OR* esperará que uma das condições seja verdadeira. Você pode combinar *AND* e *OR* na mesma expressão lógica. Quando *AND* e *OR* aparecerem na mesma cláusula *WHERE*, todos os *ANDs* serão feitos primeiros e posteriormente todos os *ORs* serão feitos.

**Tabela 05:** Operadores Lógicos.

OPERADORES	DESCRIÇÃO
<i>AND</i>	E.
<i>OR</i>	OU.
<i>NOT</i>	NÃO.

Caso queira garantir que uma determinada condição seja realizada, utilize os parênteses. Os parênteses especificam a ordem na qual os operadores devem ser avaliados (prioridade). Sempre que você estiver em dúvida sobre qual dos dois operadores será feito primeiro quando a expressão é avaliada, use sempre parênteses para definir a prioridade das expressões.

#### **AND**

operador E, utilizado em operações lógicas.

#### **OR**

operador OU, utilizado em operações lógicas.

## ■ OPERADORES ESPECIAIS

No SQL existem cinco operadores que operam com todos tipos de dados. A tabela 06 apresenta esses operadores.

**Tabela 06:** Operadores especiais para todos os tipos de dados.

OPERADOR	SIGNIFICADO
<i>BETWEEN [vl_inicial] AND [vl_final].</i>	Entre dois valores (inclusive).
<i>IN (lista).</i>	Compara uma lista de valores.
<i>LIKE.</i>	Compara um parâmetro alfanumérico.
<i>IS NULL.</i>	É um valor nulo.
<i>EXISTS.</i>	Utilizado para verificar se uma subconsulta retorna alguma linha.

**BETWEEN**  
entre dois valores.

O operador **BETWEEN** pode ser utilizado para verificar se um valor está dentro de uma faixa de valores. Por exemplo, se você quiser listar todos os países que tenham a chave primária entre 2 e 8 utilize:

```
SELECT id_pais, nm_pais  
FROM TB_PAIS  
WHERE id_pais BETWEEN 2 AND 8;
```

**IS NULL**  
valida se é nulo.

Por via de regra, o menor valor na clausula *BETWEEN* deve ser sempre listado primeiro e o maior valor deve ser listado após o *AND*. Outro operador **IS NULL** verifica quais campos em uma tabela estão com valores nulos. Assim o *SELECT* vai retornar apenas as linhas ou tuplas com o atributo nulo. Por exemplo, se quiser solicitar a lista de países que não apresentam número da população.

```
SELECT id_pais, nm_pais  
FROM TB_PAIS  
WHERE nr_populacao IS NULL;
```

**LIKE**  
compara um parâmetro alfanuméricico.

O operador especial **LIKE** é utilizado como coringa para encontrar padrões ou *strings* em um atributo. Algumas vezes você precisa procurar valores que você não conhece exatamente. Usando o operador **LIKE** é possível selecionar linhas combinando parâmetros alfanuméricos. O caractere de porcentagem ou % é utilizado como coringa nas pesquisas de *strings*. Também é possível utilizar o underscore ou simplesmente \_ para criar o coringa de um caractere. Experimente o comando:

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE 'A%';
```

No exemplo, todos os países que começam com a letra A em seu nome, serão listados. Você pode pedir que se procure uma parte da *string*, como apresentado a seguir:

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE 'Br%';
```

Ou pode solicitar que tenha o final de acordo com a regra da consulta. O comando a seguir utiliza o coringa na frente da *string* fazendo com que o final seja observado e esteja de acordo com a condição.

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE '%ia';
```

Outra opção é colocar o coringa antes e depois de uma sequência de caracteres, fazendo com que apenas as tuplas que tiverem parte da *string* no meio dela sejam retornadas. O comando a seguir apresenta um caso semelhante de uso.

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE '%ema%';
```

A utilização do *underscore* tem a mesma sintaxe do percentual; contudo, apenas um caractere será substituído. O exemplo seguinte irá retornar todas as linhas que podem satisfazer a condição \_rásil.

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE '_rásil';
```

Lembre-se que você pode utilizar combinações livremente, apresentando em um único *LIKE* \_ e % simultaneamente.

```
SELECT nm_pais  
FROM TB_PAIS  
WHERE nm_pais LIKE '%a_ca';
```

Lembre-se que a maioria dos SGBDs implementa o SQL como não sensitive-case mas diferencia os caracteres dos dados armazenados. Isso quer dizer que BRASIL, Brasil e brasil tem caracteres ASCII dife-

rentes e retornam valores diferentes. Para evitar problemas, você pode criar uma regra de restrição que pode exigir uma determinada forma ao inserir os dados. Em outra aula, você verá uma outra forma de solucionar este problema, transformando momentaneamente a cadeia de caracteres em maiúsculas ou minúsculas. O operador especial *IN* cria uma lista de valores. Seu uso é semelhante ao operador *OR*, sendo que todos os valores contidos na lista do operador *IN* devem ser do mesmo tipo de dados. Cada um dos valores da lista será testado e comparado ao atributo, se o valor corresponder a qualquer valor da lista, a tupla é selecionada.

```
SELECT nm_pais, id_pais  
FROM TB_PAIS  
WHERE id_pais IN (1,4,8,10);
```

Outra forma de realizar a mesma seleção

```
SELECT nm_pais, id_pais  
FROM TB_PAIS  
WHERE nm_pais IN ('Brasil','Países Baixos','Reino Unido','França');
```

O operador especial *IN* receberá maior atenção quando você for aprender exclusivamente sobre as subconsultas. Por fim, o operador especial **EXISTS** é útil para criar condições onde se deseja saber se existem dados em uma outra consulta. Ou seja, se a subconsulta retornar qualquer tupla, então a consulta principal será executada.

```
SELECT nm_piloto, id_pais  
FROM TB_PILOTO  
WHERE EXISTS (  
    SELECT *  
    FROM TB_PAIS  
    WHERE id_pais=3  
) ;
```

A subconsulta será executada primeira e como ela retorna uma tupla ou linha, a consulta principal pode ser executada. Como o operador *IN*, o operador **EXISTS** é utilizado com subconsultas que serão vistos em outra aula.

Retornando a sintaxe do *SELECT*, você notará que falta ver a cláusula **ORDER BY**. Ela serve para ordenar uma lista de forma ordenada.

#### ORDER BY

coloca ou ordena uma lista.

Normalmente, a ordem das linhas retornadas de uma pesquisa é indefinida. A cláusula *ORDER BY* pode ser usada para ordenar as linhas. Se você for usar o comando *ORDER BY*, lembre-se que ele deve ser sempre a última instrução da cláusula de declaração de um *SELECT*. Por padrão, a ordem da instrução *ORDER BY* é sempre crescente, mas você pode solicitar a ordem decrescente utilizando **DESC**. Não é necessário definir a ordem crescente, mas se você quiser, a instrução **ASC** declara de forma explícita. Para testar a instrução, digite:

### DESC

descreve a estrutura de uma tabela.

```
SELECT nm_pais, id_pais  
FROM TB_PAIS  
ORDER BY nm_pais;
```

Outra forma seria:

```
SELECT nm_pais, id_pais  
FROM TB_PAIS  
ORDER BY nm_pais ASC;
```

Se quiser a ordem decrescente, utilize:

```
SELECT nm_pais, id_pais  
FROM TB_PAIS  
ORDER BY nm_pais DESC;
```

Para avaliar a ordenação de múltiplas colunas, você deve inserir mais alguns pilotos. Assim digite:

```
INSERT INTO TB_PILOTO VALUES (2,'JOLYON PALMER',  
TO_DATE('20/01/1991', 'dd/mm/yyyy'),  
8,'M',1  
)  
  
INSERT INTO TB_EQUIPE VALUES (2,'Ferrari',13);  
  
INSERT INTO TB_PILOTO VALUES (3,'SEBASTIAN VETTEL',  
TO_DATE('03/07/1987', 'dd/mm/yyyy'),  
2,'M',2  
)
```

Agora com três pilotos, você pode avaliar a ordenação de múltiplas colunas. É possível utilizar mais de uma coluna na cláusula *ORDER BY*. O limite de colunas é o número de colunas da tabela. Na cláusula *ORDER BY*

especifica-se as colunas que serão ordenadas, separando-as por vírgula. Se algumas ou todas serão invertidas, especifique *DESC* depois de cada uma das colunas. Para ordenar por duas colunas, digite-o com o comando:

```
SELECT id_equipe, nm_piloto
FROM TB_PILOTO
ORDER BY id_equipe, nm_piloto;
```

Neste caso, o *id\_equipe* será o primeiro campo ordenado e quando ocorrer repetição de dados, a segunda coluna será ordenada. Como no exemplo existem dois pilotos na mesma equipe, será ordenado primeiro as equipes e depois os nomes dos pilotos. Outra forma de escrever o mesmo comando é substituir o nome das colunas que devem ser ordenadas pelo número ou ordem em que aparecem no retorno de dados. Veja a instrução equivalente a anterior:

```
SELECT id_equipe, nm_piloto
FROM TB_PILOTO
ORDER BY 1, 2;
```

A figura 04 apresenta o resultado do *SELECT*. Esta notação é bem útil quando a definição das colunas é muito extensa ou apresenta fórmulas e cálculos complexos.

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Conexões' (Connections) panel shows a connection named 'Aluno'. The central area contains a 'Planilha' (Worksheet) tab with the following SQL code:

```
select nm_pais, id_pais
from tb_pais
order by nm_pais desc;

insert into tb_piloto values (2,'JOLYON PALMER',TO_DATE('20/01/1991','DD/MM/YYYY'),8,'M',1);
INSERT INTO TB_EQUIPE VALUES (2,'Ferrari',13);
insert into tb_piloto values (3,'SEBASTIAN VETTEL',TO_DATE('03/07/1987','DD/MM/YYYY'),2,'M',2);

select id_equipe, nm_piloto
from tb_piloto
order by 1,2;
```

Below the worksheet, the 'Saída do Script' (Script Output) and 'Resultado da Consulta' (Query Result) panes are visible. The 'Resultado da Consulta' pane displays the following data:

ID_EQUIPE	NM_PILOTO
1	JOLYON PALMER
2	NICO HULKENBERG
3	SEBASTIAN VETTEL

**Figura 04:** Resultado do *SELECT* com *ORDER BY*.

A cláusula *ORDER BY* é usada na pesquisa quando você quer mostrar as linhas em uma ordem específica. Sem a cláusula *ORDER BY* as linhas são retornadas na ordem conveniente para o SGBD. Lembre-se que esse comando não altera a ordem dos dados que estão armazenados no banco de dados.

Além da ordenação, o comando *SELECT* pode solicitar dados distintos ou diferentes. A cláusula **DISTINCT** produz uma lista de valores distintos, ou seja, sem repetições. Por exemplo:

```
SELECT id_equipe  
FROM TB_PILOTO;
```

### DISTINCT

retorna dados  
distintos.

Retorna os dados apresentados na tabela 07.

**Tabela 07:** Todos os códigos de equipe na tabela TB\_PILOTO.

ID_EQUIPE
1
1
2

Observe se utilizar o comando:

```
SELECT DISTINCT id_equipe  
FROM TB_PILOTO;
```

O retorno dos dados é diferente, pois serão filtradas as linhas duplicadas, conforme apresentado na tabela 08.

**Tabela 08:** Código de equipe distintos da tabela TB\_PILOTO.

ID_EQUIPE
1
2

Finalizando a sessão no SQL, todas as expressões podem ser negativas com o uso do comando *NOT*. Com ele você pode criar expressões como *NOT BETWEEN* para indicar tudo que estiver fora da faixa, *NOT IN* para indicar tudo que não estiver na lista, *NOT LIKE* para indicar tudo que não contiver a linha de caracteres e *IS NOT NULL* indicando tudo que não for nulo.

# ■ UPDATE E DELETE

## UPDATE

comando para atualizar dados.

Para atualizar, alterar ou modificar os dados em uma tabela, você irá utilizar o comando **UPDATE**. A sintaxe desse comando é:

```
UPDATE nome_da_tabela  
SET nome_da_coluna = novo_valor [, nome_da_coluna = novo_valo-  
lor]  
[WHERE lista_de_condição];
```

Por exemplo, você pode inserir os dados de alguns pilotos com o código SQL abaixo:

```
INSERT INTO TB_EQUIPE VALUES (3, ' McLaren-Honda', 8);  
  
INSERT INTO TB_EQUIPE VALUES (4, ' Williams-Mercedes', 8);  
  
INSERT INTO TB_PILOTO VALUES (4, ' KIMI RÄIKKÖNEN',  
TO_DATE('17/10/1979', 'dd/mm/yyyy'),  
2, 'M', 2  
);  
  
INSERT INTO TB_PILOTO VALUES (5, ' STOFFEL VANDOORNE',  
TO_DATE('03/07/1987', 'dd/mm/yyyy'),  
2, 'M', 3  
);  
  
INSERT INTO TB_PILOTO VALUES (6, ' FERNANDO ALONSO',  
TO_DATE('29/07/1981', 'dd/mm/yyyy'),  
7, 'M', 3  
);
```

Após a inserção dos dados, note que o país de origem está errado. Assim para corrigir:

```
UPDATE TB_PILOTO  
SET id_pais = 3  
WHERE id_piloto = 4;
```

E se precisar corrigir mais de um tributo pode utilizar:

```
UPDATE      TB_PILOTO
SET          dt_nascimento = TO_DATE('26/03/1992', 'dd/mm/
yyyy'),
            id_pais = 6
WHERE        id_piloto = 5;
```

Observe que é importante definir muito bem a cláusula *WHERE*, pois a ausência desta cláusula provocaria alterações em todos os dados armazenados na tabela TB\_PILOTO. Portanto, tome cuidado e sempre especifique a cláusula *WHERE* quando for utilizar o comando *UPDATE* para não alterar sem querer todas as linhas da tabela especificada. Sempre confirme as correções utilizando o comando *SELECT*. Para verificar se o comando *UPDATE* não afetou todas as linhas da tabela TB\_PILOTO, digite:

```
SELECT * FROM TB_PILOTO;
```

É fácil excluir uma linha ou tupla da tabela. Para apagar os dados em uma tabela você utiliza o comando **DELETE** que apresenta a seguinte sintaxe:

```
DELETE FROM nome_da_tabela
[WHERE lista_de_condição];
```

### DELETE

comando para apa-  
gar dados.

Para exemplificar, você pode inserir dados na TB\_PROVA para apagar em seguida. Utilize o comando:

```
INSERT INTO TB_PROVA VALUES (1,TO_DATE('12/11/2017','dd/mm/
yyyy'),'cancelado');
INSERT INTO TB_PROVA VALUES (2,TO_DATE('26/03/2017','dd/mm/
yyyy'),'realizado');
```

Agora que existem duas linhas, você poderá apagar uma com o comando:

```
DELETE FROM TB_PROVA
WHERE id_prova = 2;
```

O comando *DELETE* irá apagar a prova de 26 de março de 2017. É importante você identificar corretamente qual tupla deseja apagar, pois se mais de uma tupla satisfizer a condição, esta também será apagada.

Por isso, no exemplo foi utilizada a chave primária para identificar de forma inequívoca, ou seja, sem erros a tupla desejada. Caso você queira apagar todas as linhas de uma tabela, mas manter sua estrutura, digite:

```
DELETE FROM TB_PROVA;
```

Este comando deve ser utilizado com cuidado, pois apaga todas as linhas da tabela. Antes de utilizar o comando *DELETE*, confirme a operação visualizando as linhas que serão atualizadas com o comando *SELECT*, pois ambos apresentam a mesma condição *WHERE*. Nunca omita a cláusula *WHERE*. No caso da omissão, todos os registros da tabela serão eliminados.

#### **CREATE TABLE AS**

cria uma tabela com base em uma consulta.

#### **RENAME**

permite trocar o nome de uma tabela.

#### **COMMIT**

grava todas as alterações.

#### **ROLLBACK**

desfaz quaisquer alterações.

O comando ***CREATE TABLE AS SELECT*** cria uma tabela com sua estrutura e dados tendo como base um comando *SELECT*.

```
CREATE TABLE TB_PAIS  
AS SELECT * FROM TB_TEMP;
```

Outro comando que você pode precisar é o ***RENAME***. Este comando é utilizado nos casos de alteração de nome das tabelas. Sua sintaxe é bem simples:

```
RENAME nome_antigo_tabela TO nome_novo_tabela;
```

Para exemplificar

```
RENAME TB_TEMP TO TB_PAIS_BK;
```

Outro comando interessante presente na linguagem SQL é a capacidade que dispomos de cancelar ou gravar uma série ações. Depois que você inicia uma sessão, o SGBD cria um ponto de referência para executar uma sequência de ações no SQL. Uma transação é um conjunto de operações delimitadas por um início e um fim. Iniciando quando se executa o primeiro comando SQL e terminando de acordo com a situação. Os comandos ***COMMIT*** e ***ROLLBACK*** são responsáveis por nos ajudar nesta tarefa. O comando *COMMIT*, quando executado, grava todas as alterações no banco de dados ou os efeitos dos comandos de uma transação como *INSERT*, *DELETE* e *UPDATE*. A sintaxe é simples para o comando *COMMIT*:

```
COMMIT;
```

Quando você cria uma sessão, todas as suas alterações só estão visíveis para você enquanto estiver utilizando a *interface*. Se você quiser tornar perene suas alterações, ou visíveis para outros usuários, você deve utilizar o comando *COMMIT*. É interessante notar que quando você sai de uma sessão e a mesma é encerrada sem problemas, ocorre um *COMMIT* implícito. Se sua sessão finaliza por qualquer outro problema, ocorre um *ROLLBACK* implícito. Além disso, todo comando *DDL* tal como *CREATE*, *ALTER* e *DROP* e *DCL* como *GRANT* e *REVOKE* provocam o fim da transação corrente, havendo um *COMMIT* implícito.

Se o comando *COMMIT* não foi utilizado para armazenar de forma perene as alterações, é possível restaurar o banco de dados até seu último ponto de *COMMIT* ou *ROLLBACK*. O comando *ROLLBACK* desfaz quaisquer alterações desde o último *COMMIT* e retorna os dados aos valores existentes antes das alterações. A sintaxe para o comando *ROLLBACK* é bem simples:

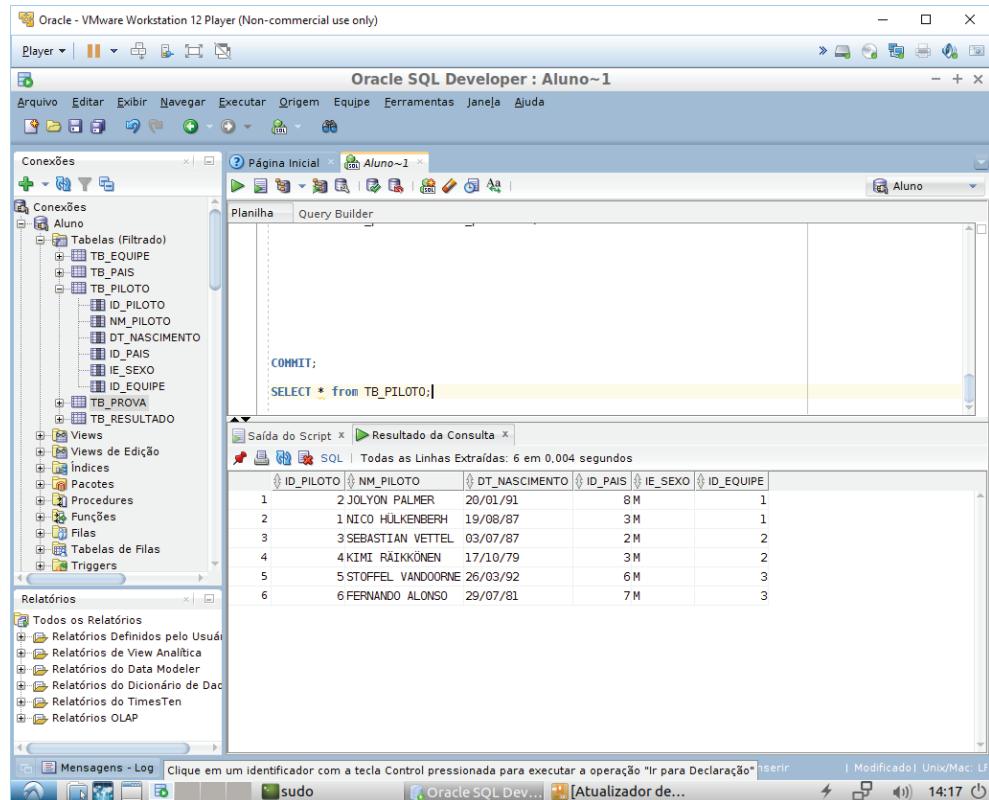
```
ROLLBACK;
```

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados que garante por si mesmo a integridade das relações existentes entre as tabelas e seus índices. Alguns SGBD aplicam *COMMIT* implícito a diversos comandos, de forma que você deve consultar no manual do SGBD quais são os comandos. Para exemplificar, experimente os seguintes comandos;

```
COMMIT; --inicia um ponto de salvamento
```

```
SELECT * FROM TB_PILOTO;
```

A figura 05 apresenta o resultado da consulta a TB\_PILOTO.



**Figura 05:** Resultado da consulta `SELECT * FROM TB_PILOTO`.

Para avaliar a capacidade de desfazer comandos *DML*, utilize o código para inserir um novo piloto.

```
INSERT INTO TB_PILOTO VALUES (
7, 'FELIPE MASSA', TO_DATE ('25/04/1981', 'dd/mm/yyyy'), 1, 'M', 4
);
```

```
SELECT * FROM TB_PILOTO;
```

Observe como sai o resultado da consulta após a inserção realizada com sucesso na figura 06.

The screenshot shows the Oracle SQL Developer interface. In the central workspace, a query builder window displays the following SQL code:

```

COMMIT;
SELECT * from TB_PILOTO;
INSERT INTO TB_PILOTO VALUES (7, 'FELIPE MASSA', TO_DATE('25/04/1981', 'DD/MM/YYYY'), 1, 'M', 4);
SELECT * from TB_PILOTO;

```

Below the code, the "Saída do Script" (Script Output) tab shows the results of the last SELECT statement, which returns 7 rows of data from the TB\_PILOTO table. The columns are ID\_PILOTO, NM\_PILOTO, DT\_NASCIMENTO, ID\_PAIS, IE\_SEXO, and ID\_EQUIPE. The data is as follows:

ID_PILOTO	NM_PILOTO	DT_NASCIMENTO	ID_PAIS	IE_SEXO	ID_EQUIPE
1	ZOLYON PALMER	20/01/91	8 M	1	
2	NICO HULKENBERG	19/08/87	3 M	1	
3	SEBASTIAN VETTEL	03/07/87	2 M	2	
4	KIMI RAIKKÖNEN	17/10/79	3 M	2	
5	STOFFEL VANDOORNE	26/03/92	6 M	3	
6	FERNANDO ALONSO	29/07/81	7 M	3	
7	FELIPE MASSA	25/04/81	1 M	4	

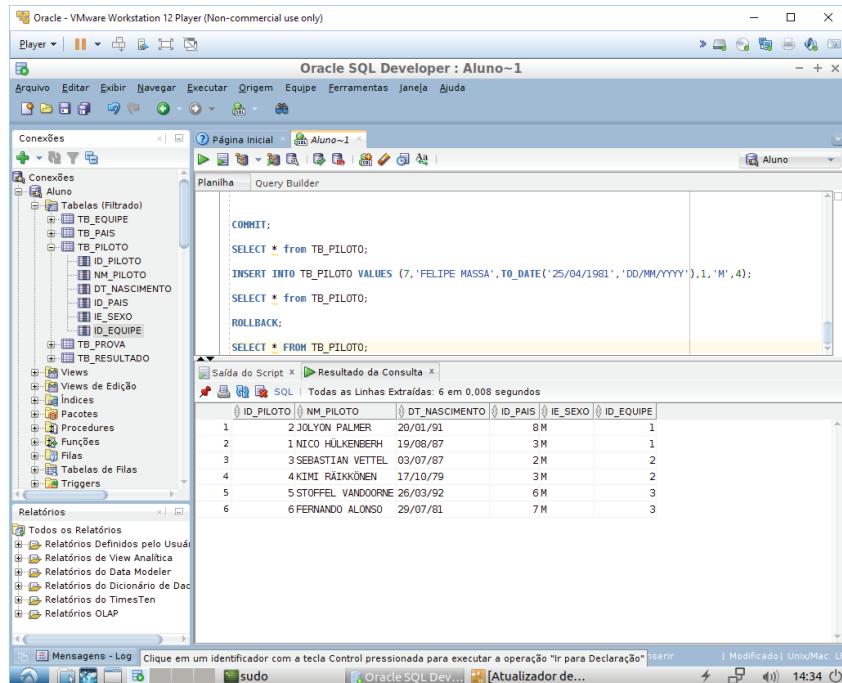
**Figura 06:** Resultado da consulta `SELECT` após o `INSERT`.

Em seguida digite:

`ROLLBACK;`

`SELECT * FROM TB_PILOTO;`

Observe os dados que a seleção retorna após o comando `ROLLBACK` na figura 07.



**Figura 07: Retorno de dados após ROLLBACK.**

Os dados após o comando *ROLLBACK* voltam ao seu estado no último *COMMIT* válido, cancelando todas as operações que ocorreram entre o *COMMIT* e o *ROLLBACK*. Se você accidentalmente esquecesse de colocar a cláusula *WHERE* em um comando *DELETE*, você pode desfazer as ações. Experimente a sequência e avalie os resultados do *SELECT*.

```

COMMIT;

SELECT * FROM TB_PILOTO;

DELETE FROM TB_PILOTO; --Erro que apagará todas as linhas da tabela

SELECT * FROM TB_PILOTO;

ROLLBACK;

SELECT * FROM TB_PILOTO;

```

Os comandos *COMMIT* e *ROLLBACK* são capazes de efetivar ou cancelar tudo ou nada, não existe meio termo nestes comandos. Se for escrever um bloco com várias instruções a serem executadas, a série inteira pode ser salva ou desfeita como um grande grupo de co-

mandos. A declaração **SAVEPOINT** faz parte dos comandos *COMMIT* e *ROLLBACK*. Este comando estabelece demarcações ou pontos dentro de uma transação. Seu objetivo é permitir que os comandos *COMMIT* ou *ROLLBACK* possam se subdividir e que alguns pontos possam ser salvos ou ter a transação desfeita. Existem algumas regras para se utilizar o **SAVEPOINT**. A primeira é que todas as declarações **SAVEPOINT** devem incluir um nome. Cuidado para não duplicar os nomes dos **SAVEPOINT**, pois se você utilizar o mesmo nome, a marcação não acontece um erro e sim uma mudança do **SAVEPOINT** anterior para a nova posição, o que tornaria impossível recuperar e apagar aquele ponto. A terceira regra é que uma vez confirmada com *COMMIT* um grupo de transações, todos os pontos de salvamento existentes serão apagados da memória e quaisquer referências a eles após o *COMMIT* implicará erro. A sintaxe do comando é simples e segue a estrutura:

```
SAVEPOINT nome_ponto;
```

Para exemplificar, experimente:

```
SAVEPOINT sp_1;

INSERT INTO TB_PILOTO VALUES (
7, 'FELIPE MASSA', TO_DATE('25/04/1981', 'dd/mm/yyyy'), 1, 'M', 4
);

SELECT * FROM TB_PILOTO;

SAVEPOINT sp_2;

INSERT INTO TB_PILOTO VALUES (
8, 'LANCE STROLL', TO_DATE('29/10/1990', 'dd/mm/
yyyy'), 12, 'M', 4
);

SELECT * FROM TB_PILOTO;

ROLLBACK TO sp_2;

SELECT * FROM TB_PILOTO;
```

O **SAVEPOINT** é útil na gestão de uma grande série de transações em que a validação incremental é exigida. Pode lhe ajudar a corrigir e validar erros em partes das instruções antes de validar todas as ações.

## SAVEPOINT

pontos ou marcações dentro de uma transação.



Nesta sexta aula, você aprendeu como alterar tabelas com o comando SQL *ALTER TABLE*. Viu como funciona a inserção de dados com o ***INSERT INTO***. Aprendeu a alterar e excluir registros de dados com *UPDATE* e *DELETE*. Vimos a sintaxe completa da instrução *SELECT* e você pode experimentar os comandos básicos do *SELECT*. Além disso, você pode aprender sobre os operadores *BETWEEN*, *IN*, *LIKE* e *IS NULL*.

## INSERT INTO

comando SQL para inserir dados em uma tabela.

## Atividades

01. Com base na tabela gere o script que insere os dados da TB\_EQUIPE

EQUIPE	
Ferrari	Itália
Force India-Mercedes	Índia
Haas-Ferrari	Estados Unidos
McLaren-Honda	Reino Unido
Mercedes	Alemanha
Red Bull-TAG Heuer	Áustria
Renault	França
Sauber-Ferrari	Suíça
Toro Rosso	Itália
Williams-Mercedes	Reino Unido

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**02.** Faça uma pesquisa e complete a TB\_PILOTO com os pilotos do calendário de 2017.

## Anotações



# AULA 7

## MANIPULAÇÃO DE DADOS COM SQL

### NESTA AULA

- » QUERY e SUBQUERY
- » Agrupando dados
- » Funções de data.

### METAS DE COMPREENSÃO

- » Conhecer as subconsultas.
- » Aprender a utilizar subconsultas para resolver problemas.
- » Aprender como trabalhar com retorno de dados simples e com múltiplas linhas em uma subconsulta.
- » Aprender como agrupar dados e como utilizar as funções de grupo.

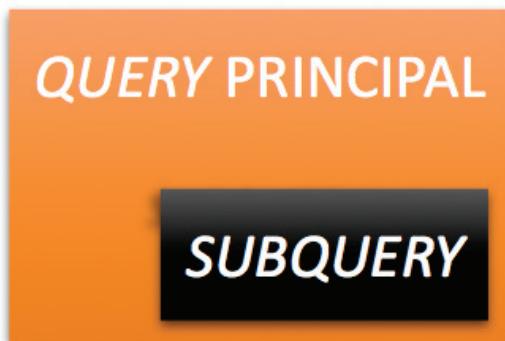
### APRESENTAÇÃO

Nesta aula você vai aprender a utilizar as subconsultas no SQL. Conhecer o conceito de SUBQUERY vai lhe ajudar a criar pesquisas complexas. Você vai aperfeiçoar o comando SELECT trabalhando com o conceito avançado de pesquisa.

Para que você possa entender estes conceitos, ensinaremos em que constitui as subconsultas, como agrupar dados e quais as funções de data.

# ■ QUERY E SUBQUERY

Nesta aula você vai aprender que uma *SUBQUERY* mais é do que uma *QUERY* ou consulta dentro de outra. As *SUBQUERY* recebem diferentes nomes como subconsultas ou *INNER QUERY* na literatura. As *SUBQUERY* podem ser utilizadas para quebrar uma consulta complexa em uma série de passos lógicos, mais fáceis de se entender. As *SUBQUERY* normalmente são expressas entre parênteses. A primeira consulta no comando SQL é conhecida como consulta externa ou *OUTER QUERY*. A consulta no interior do comando SQL é conhecida como consulta interna, *SUBQUERY* ou *INNER QUERY*. A consulta *SUBQUERY* é executada primeiro e sua saída é utilizada como entrada na *OUTER QUERY*. A *SUBQUERY* tem como base a utilização do comando *SELECT* para retornar um ou mais valores a outra consulta. O conceito pode ser representado graficamente como ilustrado na figura 01.



**Figura 01:** Conceito gráfico da *SUBQUERY*.

As subconsultas apresentam uma ampla faixa de utilização, por exemplo, você pode aplicar *SUBQUERY* em todos os comandos *DML* como *INSERT*, *UPDATE* e *DELETE* ou em lugares em que se espera um valor ou uma lista de valores. A sintaxe da *SUBQUERY* mais comum é:

```
SELECT nome_da_coluna
FROM nome_da_tabela
WHERE expressão operador (
    SELECT coluna_expressão
    FROM nome_da_tabela
);
```

Esta *QUERY* utiliza uma *SUBQUERY* interna ao lado direito da expressão de comparação *WHERE*. Para exemplificar, imagine que você queira saber o nome do piloto que nasceu no país Brasil, você poderia executar duas pesquisas:

```
SELECT id_pais  
FROM TB_PAIS  
WHERE nm_pais = 'Brasil';
```

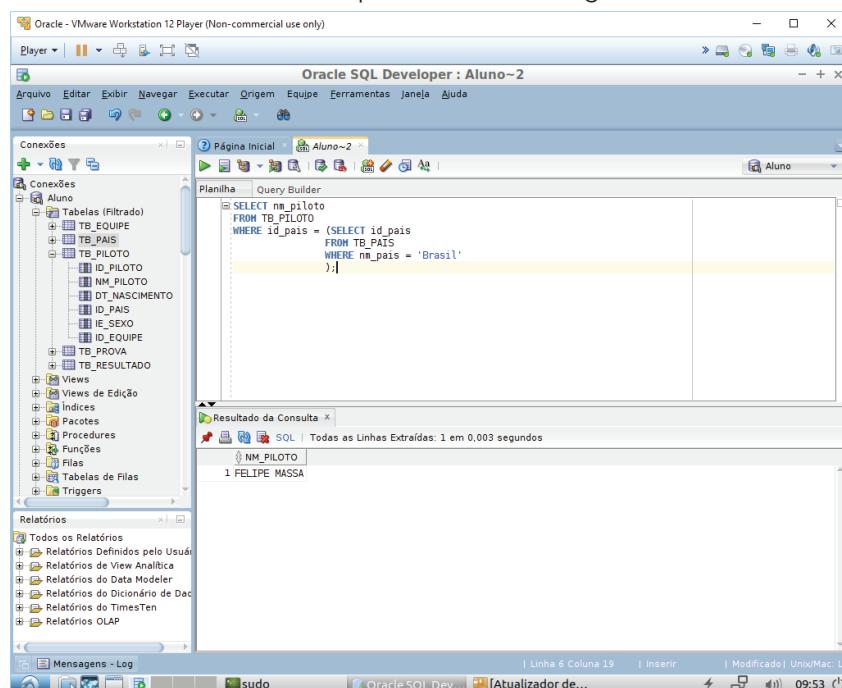
Anotar o resultado da *QUERY* e fazer uma outra pesquisa.

```
SELECT nm_piloto  
FROM TB_PILOTO  
WHERE id_pais = 1;
```

A *SUBQUERY* realiza as duas consultas de uma forma direta e bem mais elegante. A SQL ficaria assim:

```
SELECT nm_piloto  
FROM TB_PILOTO  
WHERE id_pais = (SELECT id_pais  
FROM TB_PAIS  
WHERE nm_pais = 'Brasil'  
) ;
```

O resultado da consulta é apresentado na figura 02.



**Figura 02:** Resultado da consulta com *SUBQUERY*.

## SINGLE ROW

retorno de apenas uma linha ou um registro.

Observe que esse tipo de subconsulta retorna apenas uma linha, ou seja, só existe um código ou id\_pais para o Brasil. Por isso esta *INNER QUERY* é classificada como **SINGLE ROW** visto que seu retorno é apenas uma única linha. As *SUBQUERY SINGLE ROW* podem utilizar operadores condicionais  $>$ ,  $<$ ,  $=$ ,  $\geq$  ou  $\leq$ . O valor que a subconsulta gera deve ser do mesmo tipo de dados definido antes do operador condicional. No exemplo o id\_pais da TB\_PILOTO é um *NUMBER(5)* e o dado que a subconsulta retorna deve ser do mesmo tipo de dados. Como id\_pais da TB\_PAIS é um *NUMBER(5)*, os dados da subconsulta podem ser comparados com os da consulta principal. É importante ressaltar que se a *SUBQUERY* retornar mais de uma linha ou dados de tipo diferente ao da *QUERY*, o SGBD vai gerar um erro. Além disso, as *SUBQUERY* podem trabalhar aninhada, ou seja, uma pode conter outras.

```
SELECT nm_equipe
FROM TB_EQUIPE
WHERE id_equipe = (SELECT id_equipe
                    FROM TB_PILOTO
                    WHERE id_pais = (SELECT id_pais
                                      FROM TB_PAIS
                                      WHERE nm_pais = 'Brasil')
                    );
;
```

Neste exemplo primeiro será realizada a pesquisa do id\_pais do país com o nome Brasil na TB\_PAIS. Quando a consulta retornar o valor, é realizada a consulta do id\_equipe na TB\_PILOTO com o id\_pais = 1. Após a pesquisa, então a consulta principal será processada, pesquisando o nome da equipe do piloto Felipe Massa. Contudo, muitas vezes as *SUBQUERY* vão retornar múltiplas linhas ou **MULTIPLE ROW**. Quando se deseja comparar um atributo com uma lista de valores, utiliza-se o operador *IN*. No próximo exemplo, você solicita o nome de todos os países que tenham um piloto apresentando o nome dos países em ordem alfabética.

```
SELECT DISTINCT nm_pais
FROM TB_PAIS
WHERE id_pais IN ( SELECT id_pais
                    FROM TB_PILOTO
                    )
ORDER BY nm_pais;
```

## MULTIPLE ROW

retorno de duas ou mais linhas ou registros.

Observe que podemos listar os países que participam da F1 mas não tem pilotos apenas negando a SUBQUERY.

```
SELECT DISTINCT nm_pais
FROM TB_PAIS
WHERE id_pais NOT IN (SELECT id_pais FROM TB_PILOTO)
ORDER BY nm_pais;
```

Os operadores para *INNER QUERY* com *MULTIPLE ROW* são apresentados na tabela 01.

**Tabela 01:** Operadores utilizados em *SUBQUERY* com múltiplas linhas.

OPERADOR	UTILIZAÇÃO
<i>IN</i>	É igual a qualquer membro da lista.
<i>ANY</i>	Compara o valor com cada valor retornado pela <i>SUBQUERY</i> .
<i>ALL</i>	Compara o valor com todos os valores retornados pela <i>SUBQUERY</i> .
<i>EXISTS</i>	Testa se um valor existe.

Até o momento você utilizou o operador *IN* para lidar com *INNER QUERY* que retornam mais de uma linha. Mas o operador *IN* é similar ao operador igual, ou seja, ele produz uma comparação de igualdade, retornando as linhas que correspondem ou são iguais à apresentada na lista. Às vezes, você vai precisar comparar maior ou menor que a lista, para isso existem dois operadores. O operador *ANY* compara o valor com cada valor que a *INNER QUERY* retorna, enquanto o operador *ALL* compara o valor com todos os valores que a subconsulta retorna. Para demonstrar o funcionamento, primariamente você deve popular a *TB\_RESULTADO*. Utilize os comandos:

```
INSERT INTO TB_RESULTADO VALUES (3,1,1,2,TO_DATE('01:24:11','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (15,1,2,1,TO_DATE('01:31:10','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (16,1,3,3,TO_DATE('01:33:13','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (4,1,4,4,TO_DATE('01:46:49','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (18,1,5,5,TO_DATE('01:52:58','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (7,1,6,7,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (9,1,7,11,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (23,1,8,8,TO_DATE('02:00:00','HH:MI:SS'),NULL);
```

```

INSERT INTO TB_RESULTADO VALUES (22,1,9,9,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (10,1,10,13,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (1,1,11,11,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (20,1,12,16,TO_DATE('02:00:00','HH:MI:SS'),NULL);
INSERT INTO TB_RESULTADO VALUES (5,1,13,18,TO_DATE('02:00:00','HH:MI:SS'),NULL);

```

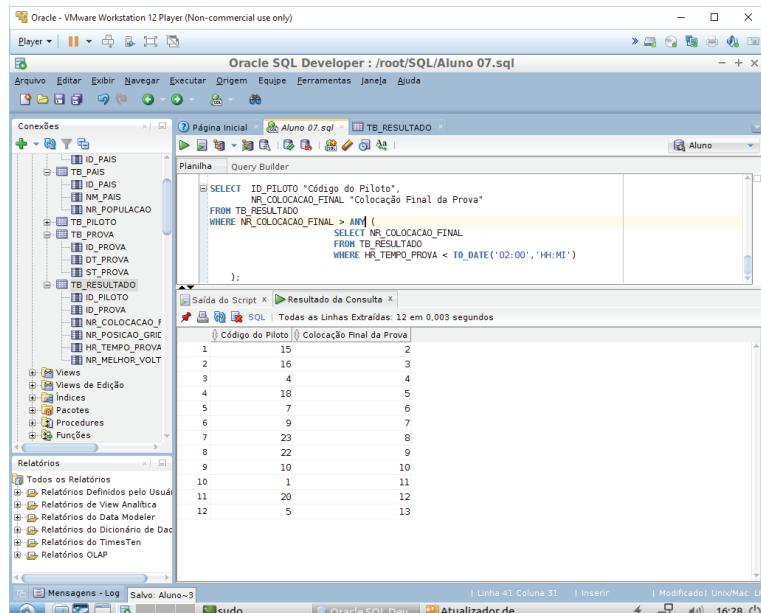
Agora que existem algumas linhas, você pode utilizar o comando:

```

SELECT      id_piloto "Código do Piloto",
nr_colocacao_final "Colocação no final da prova"
FROM        TB_RESULTADO
WHERE       nr_colocacao_final > ANY (
          SELECT nr_colocacao_final
          FROM TB_RESULTADO
          WHERE hr_tempo_prova <
to_date('02:00','HH:MI'));

```

Observe que, no exemplo, a *SUBQUERY* será a primeira a ser executada. O resultado da *SUBQUERY* será 1,2,3,4,5. Assim o SGBD passa para a *QUERY* principal procurando o código do piloto e a colocação final da prova para qualquer resultado que seja menor que qualquer elemento da lista [1,2,3,4,5]. Como as comparações podem ocorrer com qualquer elemento da lista, tem-se as 12 colocações como resultado, excluindo apenas a primeira colocação. A figura 03 apresenta o resultado da *INNER QUERY*.



**Figura 03:** Resultado da *INNER QUERY* com ANY.

Você pode solicitar que as comparações ocorram com todos ou *ALL* elementos. Utilize o código:

```
SELECT      id_piloto "Código do Piloto",
nr_colocacao_final "Colocação no final da prova"
FROM        TB_RESULTADO
WHERE       nr_colocacao_final > ALL (
          SELECT nr_colocacao_final
          FROM TB_RESULTADO
          WHERE hr_tempo_prova < to_da-
te('02:00', 'HH:MI')
);
```

Desta vez, o resultado foi diferente, pois a comparação se deu com todos os resultados da *INNER QUERY*. Observe atentamente a figura 04 e veja que o 2º, 3º, 4º e 5º colocados foram eliminados da resposta.

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Conexões' (Connections) sidebar lists several tables: ID\_PAIS, TB\_PAIS, NR\_POPULACAO, TB\_PILOTO, TB\_PROVA, and TB\_RESULTADO. The 'TB\_RESULTADO' node is expanded to show sub-tables: ID\_PILOTO, ID\_PROVA, NR\_COLOCACAO\_FINAL, NR\_POSICAO\_GRIE, NR\_TEMPO\_PROVA, and NR\_MELHOR\_VOLT. The main workspace displays the SQL query:

```
SELECT      ID_PILOTO "Código do Piloto",
NR_COLOCACAO_FINAL "Colocação Final da Prova"
FROM        TB_RESULTADO
WHERE       NR_COLOCACAO_FINAL > ALL (
          SELECT NR_COLOCACAO_FINAL
          FROM TB_RESULTADO
          WHERE NR_TEMPO_PROVA < TO_DATE('02:00', 'HH:MI')
);
```

Below the query, the 'Saída do Script' (Script Output) and 'Resultado da Consulta' (Query Result) panes are visible. The 'Resultado da Consulta' pane shows the following data:

Código do Piloto	Colocação Final da Prova
1	7
2	9
3	23
4	22
5	10
6	1
7	20
8	5

**Figura 04:** Resultado da *INNER QUERY* com *ALL*.

O resultado se deu devido a expressão *ALL* ser traduzida como compare com o maior número de colocação da lista que, neste caso, foi o 5º colocado. Uma observação importante: normalmente o uso para igualdade se dá com o operador *=* e os demais operadores aritméticos utilizam *ANY* ou *ALL*, contudo você pode utilizar o *ANY* com o igual, que será equivalente ao operador *=*.

# ■ AGRUPANDO DADOS

## GROUP BY

argumento utilizado para agrupar dados.

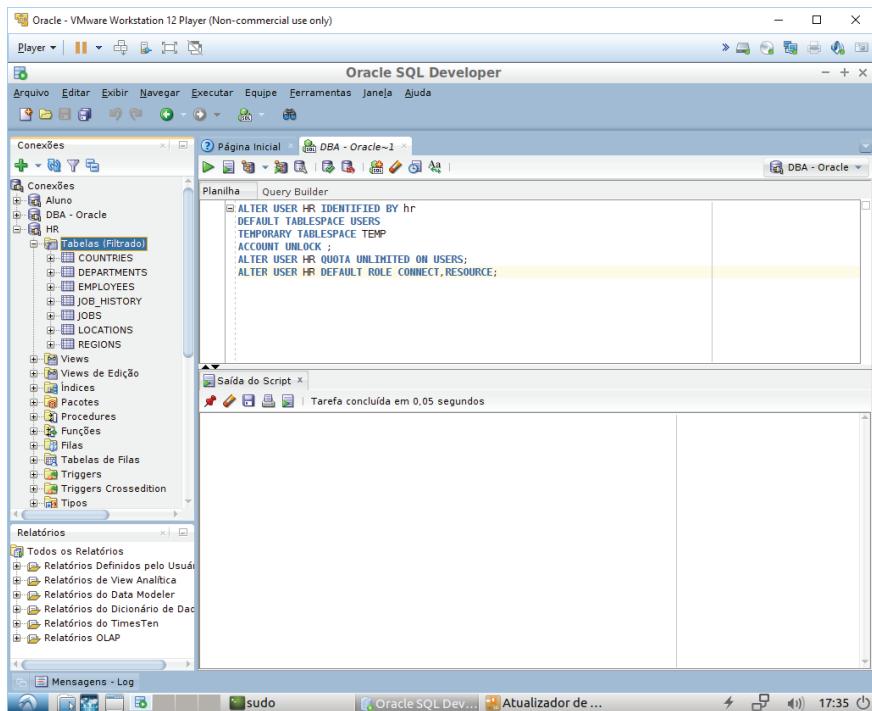
Você pode agrupar dados de maneira fácil e rápida com a instrução **GROUP BY** no comando *SELECT*. A sintaxe é:

```
SELECT [distinct] col1 [alias], coln  
FROM tab1 [alias], tabn [alias]  
WHERE condição  
GROUP BY colunas  
HAVING condição  
ORDER BY expressão ou chave [desc];
```

A cláusula *GROUP BY* exigirá que a coluna esteja listada no comando *SELECT*. Também os campos que serão exibidos ou são afetados pelo *GROUP BY* ou são funções de grupo. Assim, não solicite em uma *SELECT* atributos agrupados e atributos que não podem ser agrupados. Para facilitar a exemplificação do uso, vamos habilitar o usuário HR que foi instalado automaticamente com o SGBDR *Oracle*. Conecte como DBA – *Oracle* e digite:

```
ALTER USER HR IDENTIFIED BY hr  
DEFAULT TABLESPACE users  
TEMPORARY TABLESPACE temp  
ACCOUNT UNLOCK;  
  
ALTER USER HR QUOTA UNLIMITED ON USERS;  
  
ALTER USER HR DEFAULT ROLE CONNECT, RESOURCE;
```

Execute os comandos acima com seu usuário DBA e depois selecione Nova Conexão e a janela para criar uma nova conexão irá aparecer. No campo nome digite "HR", no campo nome do usuário digite "HR" e em senha digite "hr". Selecione o campo "Salvar Senha" e selecione o botão Testar. Se aparecer no *Status*: "Sucesso" selecione o botão Conectar. O usuário HR vem com 7 tabelas com dados já cadastrados que normalmente são utilizados nas provas de certificações da Oracle. A figura 05 apresenta a nova conexão com o SGBD com o usuário HR já desbloqueado.



**Figura 05:** Conexão com usuário HR desbloqueado.

Você pode pressionar agora <Alt><F10> simultaneamente para criar uma nova conexão. Selecione a conexão HR e uma nova aba irá se abrir. Digite:

```
SELECT first_name, last_name, salary, job_id
FROM EMPLOYEES;
```

Ao executar o comando, será exibido uma listagem com nome, sobrenome, salário e cargo dos empregados registrados na tabela *EMPLOYEES*. Se você notar, a listagem apresenta dados individuais dos empregados. Você pode agrupar os cargos para exibir com o comando *GROUP BY*.

```
SELECT job_id
FROM EMPLOYEES
GROUP BY job_id;
```

Observe que o comando agrupou as 107 linhas em apenas 19. Apesar de o agrupamento ser interessante, visto de forma isolada, ele às vezes ocorre assim. A maioria das consultas vão ocorrer com outras funções de grupo. A funcionalidade das funções de grupo é de resumir

informações, permitindo obter, por meio dos grupos de linhas, dados relevantes. As funções de grupo operam sobre um conjunto de linhas para dar um único resultado por grupo. As funções de grupos são exibidas na tabela 02.

**Tabela 02:** Funções de Grupo.

FUNÇÃO	DESCRIÇÃO
SUM	Retorna a soma de N.
AVG	Retorna a média aritmética de N.
COUNT	Retorna o número de linhas da consulta.
MAX	Retorna o valor máximo de N.
MIN	Retorna o valor mínimo de N.

Todas estas funções operam sobre um número de linhas, por exemplo, uma tabela inteira e são, portanto, funções de GRUPO. O atributo **DISTINCT** pode fazer parte de uma função de grupo para solicitar que se considere valores não duplicados. O operador **ALL** considera todos os valores e sua declaração não é necessária. Os tipos de dados dos argumentos devem ser alfanuméricos, numéricos ou *data* onde a expressão é listada. Todas as funções de grupo exceto o **COUNT(\*)** ignoram os valores nulos.

#### DISTINCT

retorna apenas valores não duplicados.

#### COUNT

serve para contar o número de valores.

#### AVG

retorna a média aritmética de um grupo de registros.

A função **AVG** retorna a média aritmética de um grupo de registros. Para calcular a média salarial dos empregados, faça:

```
SELECT AVG(salary)
FROM EMPLOYEES;
```

O resultado é apresentado na figura 06.

AVG(SALARY)
6461,831775700934579439252336448598130841

**Figura 06:** Resultado do função AVG(salary).

Ao definir a função AVG com o *GROUP BY*, você terá uma função de grupo poderosa. Experimente a seguinte instrução:

```
SELECT job_id "Cargo", AVG(salary) "Média Salarial"
FROM EMPLOYEES
GROUP BY job_id;
```

O resultado é uma lista com a média salarial de todos os cargos da empresa como apresentado na figura 07.

Cargo	Média Salarial
IT_PROG	5760
AC_MGR	12008
AC_ACCOUNT	8300
ST_MAN	7280
PU_MAN	11000
AD_ASST	4400
AD_VP	17000
SH_CLERK	3215
FI_ACCOUNT	7920
FI_MGR	12008
PU_CLERK	2780
SA_MAN	12200
MK_MAN	13000
PR_REP	10000
AD_PRES	24000
SA_REP	8350
MK_REP	6000
ST_CLERK	2785
HR_REP	6500

**Figura 07:** Listagem de cargos e média salarial.

Como a função AVG trabalha com grupos de dados, não é necessário declará-la na instrução GROUP BY.

## COUNT

A função COUNT serve para contar o número de valores não nulos de um atributo. Ela pode ser utilizada com a cláusula DISTINCT. Para exemplificar, imagine que você queira contar o número de empregados registrados na empresa.

```
SELECT COUNT(employee_id) "Número de Empregados"  
FROM EMPLOYEES;
```

A QUERY vai retornar 107 empregados como apresentado na figura 08.

Total Empregados
107

**Figura 08:** Resultado da QUERY com COUNT.

Você pode querer saber quantos empregados registrados trabalham no departamento de vendas ou Sales. Para isso utilize a instrução SQL:

```
SELECT COUNT(employee_id) "Número de Empregados"
```

```

FROM EMPLOYEES
WHERE department_id = (
    SELECT department_id
    FROM DEPARTMENTS
    WHERE department_name = 'Sales'
);

```

A consulta vai retornar 34 empregados atuando como vendedores. As consultas com *COUNT*(*nome\_da\_coluna*) sempre retorna o número de linhas com valores não nulos na coluna. Por outro lado se você quiser contar a quantidade de linhas encontradas, independente se existe ou não valores nulos em determinada coluna, você pode utilizar o *COUNT(\*)*. Quando definido como *COUNT(\*)*, você está solicitando que se conte o número de linhas encontradas na consulta, independente da presença de colunas nulas. Para exemplificar, experimente a seguinte instrução SQL:

```

SELECT COUNT(*) "Quant. Departamentos"
FROM DEPARTMENTS
WHERE location_id = (
    SELECT location_id
    FROM LOCATIONS
    WHERE city = 'Seattle'
);

```

Você pode utilizar o *COUNT* também com dados agrupados. Para exemplificar, imagine que você deseja saber a quantidade de funcionários cadastrados em cada departamento.

```

SELECT COUNT(*) AS "Quant. Empregados",
       department_id AS "Código Departamento"
FROM EMPLOYEES
GROUP BY department_id;

```

Está é uma utilização comum do comando *GROUP BY*. Lembre-se que os campos existentes na cláusula *GROUP BY* não precisam aparecer no *SELECT*. Caso queira restringir os dados, não use a cláusula *WHERE*, e sim, utilize a cláusula **HAVING**. Você não pode usar funções de grupo na cláusula *WHERE*.

### **HAVING**

equivale ao *WHERE*  
para dados  
agrupados.

## **MAX E MIN**

As funções *MAX* e *MIN* retornam o maior valor de um grupo de registro e o menor valor de um grupo de registro. Uma função de grupo pode

ser usada para procurar o maior e o menor valor em um grupo de linhas de uma tabela usando a cláusula *WHERE*. Para encontrar o menor e o maior salários dos funcionários do departamento 30, faça:

```
SELECT MIN(salary), MAX(salary)
FROM   EMPLOYEES
WHERE  department_id = 30;
```

O menor salário é 2.500 e o maior salário é 11.000. É importante você se lembrar que as funções numéricas *MAX* e *MIN* produzem apenas um valor com base em todos os valores encontrados na tabela. É comum alguns DBA esquecerem essa questão é escrever *QUERY* como essa:

```
SELECT first_name, last_name
FROM   EMPLOYEES
WHERE  salary = MAX(salary);
```

Essa instrução irá produzir um erro. Caso você queira consultar quem é o funcionário que recebe o maior salário, você deve utilizar a seguinte SQL:

```
SELECT first_name || ' ' || last_name AS "Nome", salary AS
"Salário"
FROM   EMPLOYEES
WHERE  salary = ( SELECT MAX(salary) FROM EMPLOYEES );
```

Você deve primeiro computar o maior salário entre os empregados e depois pesquisar o funcionário que recebe este salário. As funções *MAX* e *MIN* podem ser utilizadas com colunas tipo *date*. Por exemplo, você pode consultar qual o funcionário mais velho de casa com a instrução SQL:

```
SELECT first_name || ' ' || last_name AS "Nome"
FROM   EMPLOYEES
WHERE  hire_date = ( SELECT MIN(hire_date) FROM EMPLOYEES );
```

Também as funções *MAX* e *MIN* podem ser utilizadas junto com a instrução *GROUP BY* sem ser necessário declarar a função no *GROUP BY*. Por exemplo, imagine que você deseja pesquisar todos os salários que estejam acima de 10.000 por departamento.

```
SELECT department_id, MAX(salary)
FROM EMPLOYEES
GROUP BY department_id
HAVING MAX(salary)>10000;
```

## SUM

A função *SUM* calcula a soma total de um atributo específico. Para exemplificar, experimente executar a *QUERY*.

```
SELECT SUM(salary)
FROM EMPLOYEES;
```

A *QUERY* soma o total de proventos que será necessário para pagar a folha de pagamento. Você pode utilizar restrições com a cláusula *WHERE* como por exemplo:

```
SELECT SUM(salary)
FROM EMPLOYEES
WHERE department_id = 30;
```

A consulta anterior soma todos os salários dos funcionários do departamento de código 30. Também é possível calcular a soma com uma expressão. Por exemplo, calcular um aumento para os funcionários de 10%.

```
SELECT SUM(salary*1.10)
FROM EMPLOYEES;
```

A *QUERY* para cada salário acrescenta 10% sobre o valor e depois soma todas as linhas. É interessante notar que você pode excluir linhas quando estiver usando o *GROUP BY* com a cláusula *WHERE*. As linhas devem ser excluídas com a cláusula *WHERE*, antes de se criar os grupos. Para exemplificar, se você quiser mostrar a média salarial de cada cargo excluindo os departamentos 10, 20 e 30, faça:

```
SELECT department_id, AVG(salary)
FROM EMPLOYEES
WHERE department_id NOT IN (10,20,30)
GROUP BY department_id
ORDER BY department_id;
```

Observe que podem ser excluídos os departamentos 10, 20 e 30 antes da instrução *GROUP BY*. Só foi permitido o uso do *WHERE* por não restringir os dados agrupados e sim os dados antes do agrupamento. Além disso, é importante frisar que é possível agrupar dados de mais de uma coluna. Veja o exemplo:

```
SELECT department_id, job_id, SUM(salary)
FROM EMPLOYEES
GROUP BY department_id, job_id;
```

A figura 09 apresenta como os dados são agrupados visto que apresentam dois agrupamentos *department\_id* e *job\_id*.

### EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA REP	11000
80	SA REP	8600
...		
20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

20 rows selected.

**Somando os salários da tabela EMPLOYEES agrupados por departamento e cargo**

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA REP	7000

13 rows selected.

**Figura 09:** GROUP BY com múltiplas colunas.

Outro exemplo que você pode testar que apresenta o conceito, pode ser visto caso você queira mostrar a média salarial, a soma dos salários por departamento e o cargo.

```
SELECT department_id, job_id, AVG(salary), SUM(salary)
FROM EMPLOYEES
GROUP BY department_id, job_id
ORDER BY department_id;
```

# ■ FUNÇÕES DE DATA

Todos os SGBDs dão suporte a funções de data e hora. Todas as funções implementadas convertem um parâmetro quer seja data, hora ou caractere para outro formato de data, hora e caractere. A grande dificuldade é que cada fabricante de SGBD definiu um modo diferente de implementar essas instruções. A razão disso ocorrer, em parte, foi pelo fato do padrão SQL ANSI definir os tipos de dados que deveriam existir sem definir como esses dados deveriam ser tratados. As funções de data são utilizadas para manipular valores do tipo *date*. Um exemplo seria aplicar formatações para uma visualização mais refinada, ou extrair partes de uma data, as horas, dia do mês ou somente o ano. A tabela 03 apresenta algumas funções existentes.

**Tabela 03:** Funções de Data

FUNÇÃO	DESCRIÇÃO
<i>ADD_MONTHS</i>	Adiciona meses em uma determinada data.
<i>MONTHS_BETWEEN</i>	Retorna a quantidade de meses entre duas datas.
<i>NEXT_DAY</i>	Procura o próximo dia após uma data informada.
<i>LAST_DAY</i>	Retorna o último dia do mês com base em uma data informada.
<i>TRUNC</i>	Trunca uma data passada por parâmetro. Pode ser feito por dia e mês, utilizando o parâmetro <i>FMT</i> (formato).
<i>SYSDATE</i>	Retorna a data corrente com base no servidor do banco de dados.
<i>SESSIONTIMEZONE</i>	Mostra o fuso horário com base na sessão aberta no banco de dados, mediante sua localização. Vale lembrar que os fusos horários são calculados com base no meridiano de Greenwich.
<i>CURRENT_DATE</i>	Mostra a data corrente com base na zona de tempo da sessão do usuário. A zona de tempo é afetada em relação ao Meridiano. Caso não haja mudanças de zona, esta função terá o mesmo valor que <i>SYSDATE</i> mesmo que a sessão tenha sido aberta em uma zona diferente daquela em que o servidor se encontra. Já o <i>CURRENT_DATE</i> refletirá a zona onde foi aberta a sessão.

## TO\_CHAR

função para converter ou formatar um número ou data em uma string.

Para fixar, veremos cada uma dessas funções, mas antes você deve conhecer mais uma função de conversão. Você já viu a função de conversão *TO\_DATE*. Agora você deve conhecer a função **TO\_CHAR**. A fun-

ção *TO\_CHAR* tem por objetivo converter um número ou uma data para o formato *string* de caractere. Além desse fim, ela acaba sendo muito utilizada para formatação visual de dados. Para exemplificar, imagine que você quer separar o valor do salário dos empregados, utilize:

```
SELECT      last_name, TO_CHAR(
    salary,
    '9G999G999D0',
    'NLS_NUMERIC_CHARACTERS= ''.,'' ') AS "Salário"
FROM        EMPLOYEES;
```

O resultado da *QUERY* é apresentado na figura 10. Observe que as demais linhas foram suprimidas.

Nome do Funcionário	Salário
1 Steven King	24,000.00
2 Neena Kochhar	17,000.00
3 Lex De Haan	17,000.00
4 Alexander Hunold	9,000.00
5 Bruce Ernst	6,000.00

**Figura 10:** Atributo *TO\_CHAR* utilizado para formatar dados.

Foi utilizada a função *TO\_CHAR* como a finalidade de formatar os valores dos salários. O elemento de formatação para casas decimais e milhar foi informado na *string* '9G999G999D0'. Note que foi utilizado o parâmetro *NLS\_NUMERIC\_CHARACTERS*, para definir quais caracteres devem ser utilizados como separador dos elementos. Outro exemplo que você pode experimentar é:

```
SELECT 'Hoje é o dia '||TO_CHAR(SYSDATE,'DDD')||' do ano'
AS DIA
FROM DUAL;
```

Essa consulta converte a data do servidor no formato de dias corridos do ano. Após isso todos os caracteres são concatenados em uma frase. A figura 11 apresenta o resultado da *QUERY*.

DIA
1 Hoje é o dia 207 do ano

**Figura 11:** QUERY com *TO\_CHAR(data,'DDD')*.

Outro exemplo que você pode experimentar é:

```

SELECT      first_name || last_name AS NOME,
            hire_date AS "Dia da contratação",
            TO_CHAR(hire_date,'DAY') AS "Dia da Semana"
FROM        EMPLOYEES;

```

A pesquisa irá retornar o nome completo dos empregados, a data em que foram contratados e qual era o dia da semana. O resultado é apresentado na figura 12.

NOME	Dia da Contratação	Dia da Semana
1 Steven King	17/06/03	TERÇA-FEIRA
2 Neena Kochhar	21/09/05	QUARTA-FEIRA
3 Lex De Haan	13/01/01	SÁBADO
4 Alexander Hunold	03/01/06	TERÇA-FEIRA
5 Bruce Ernst	21/05/07	SEGUNDA-FEIRA
6 David Austin	25/06/05	SÁBADO
7 Valli Pataballa	05/02/06	DOMINGO
8 Diana Lorentz	07/02/07	QUARTA-FEIRA
9 Nancy Greenberg	17/08/02	SÁBADO
10 Daniel Faviet	16/08/02	SEXTA-FEIRA
...		

**Figura 12:** QUERY com TO\_CHAR(data,'DAY')

Se você precisar formatar um valor do tipo *NUMBER* para o Real, pode utilizar um código SQL semelhante a esse:

```

SELECT      first_name || last_name AS NOME,
            'R$ ' || TO_CHAR(salary,'L9G999G999D00') AS "Salário"
FROM        EMPLOYEES;

```

O resultado da consulta é apresentado na figura 13.

NOME	Salário
1 Steven King	R\$ 24.000,00
2 Neena Kochhar	R\$ 17.000,00
3 Lex De Haan	R\$ 17.000,00
4 Alexander Hunold	R\$ 9.000,00
5 Bruce Ernst	R\$ 6.000,00
6 David Austin	R\$ 4.800,00
7 Valli Pataballa	R\$ 4.800,00
8 Diana Lorentz	R\$ 4.200,00
9 Nancy Greenberg	R\$ 12.008,00
...	

**Figura 13:** QUERY com TO\_CHAR(número, 'L9G999D00').

Existem alguns elementos de formatação que são mais utilizados. A tabela 04 apresenta os que você provavelmente irá utilizar.

**Tabela 04:** Elementos de formatação.

ELEMENTO	USO
9	O nove representa um caractere que será substituído pelo caractere referente ao valor passado como parâmetro. Os zeros na frente são tratados como espaços em branco.
0	Todos os zeros iniciais ou finais são exibidos em vez de um espaço em branco.
\$	Prefixo do símbolo de moeda na primeira posição.
S	Exibe o sinal +/- no início ou no final .
D	Localização do ponto decimal. Os noves de ambos os lados refletem o número máximo de dígitos permitidos.
G	Especifica um separador de grupo como milhar com uma vírgula.
L	Especifica uso da moeda local.
,	Coloca uma vírgula na posição indicada.
.	Coloca o ponto decimal, independentemente do separador decimal.
fm	Remove os espaços em branco no início e no final.
mi	Colocado no final da máscara irá definir o sinal de subtração à direita nos valores negativos.
PR	Define que valores negativos serão exibidos entre colchetes.
EEEE	Notação científica.
U	Utiliza o símbolo monetário Euro.
B	Exibe valores zero em branco.
YYYY ou YYY ou YY ou Y	Últimos 4, 3, 2 ou 1 digito(s) do ano.
Q	Um quarto de ano.
MM	Exibe o mês.
MONTH	Exibe o nome do mês.
MON	Exibe o nome do mês com 3 letras abreviadas.
WW	Semana do ano.
DDD	Exibe o dia do ano.
DD	Exibe o dia do mês.
DAY	Exibe o nome do dia da semana.
DY	Exibe o nome do dia da semana com 3 letras abreviadas.
AM ou PM	Indicador meridiano.
HH ou HH12	Define horas do dia de 0-12.
HH24	Define horas do dia de 0-23 .
MI	Define minutos.
SS	Segundos.
SSSSS	Segundos passado meia-noite de 0-86399.



você sabia?

Nas instruções SQL, você pode especificar qual o modelo e o formato dos argumentos para as funções *TO\_CHAR* e *TO\_DATE*. Você pode ler e ter acesso a todos os argumentos que seu banco de dados oferece. Acesse o manual do SGBD em: [https://docs.oracle.com/cd/B19306\\_01/server.102/b14200/sql\\_elements004.htm](https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm)  
Acessado em 26/07/2017

O próximo exemplo utiliza o *COUNT* e o *TO\_CHAR* para pesquisar quantos funcionários foram contratados em cada mês.

```
SELECT COUNT(*) AS "Quant. Contratados",
       TO_CHAR(hire_date, 'MM') AS "Mês"
  FROM EMPLOYEES
 GROUP BY TO_CHAR(hire_date, 'MM')
 ORDER BY 2;
```

Na pesquisa acima, utilizando o *TO\_CHAR*, todos os dados são agrupados no formato numérico. Então o *COUNT* conta quantos funcionários foram contratados em cada mês. O resultado da consulta é exibido na figura 14.

	Quant. Contratados	Mês
1	14	01
2	13	02
3	17	03
4	7	04
5	6	05
6	11	06
7	7	07
8	9	08
9	5	09
10	6	10
11	5	11
12	7	12

**Figura 14:** Resultado da QUERY com COUNT e TO\_CHAR.

Outro exemplo seria escrever a data por extenso. Experimente o código SQL:

```
SELECT 'São Paulo, ' || TO_CHAR(SYSDATE, 'DD') || ' de ' ||
       INITCAP(TO_CHAR(SYSDATE, 'fmMONTH')) || ' de ' ||
```

```
TO_CHAR(SYSDATE, 'YYYY') || '.' AS "Data"  
FROM DUAL;
```

O resultado é apresentado na figura 15.

Data
1 São Paulo, 26 de Julho de 2017.

**Figura 15:** Resultado da QUERY com TO\_CHAR.

Ou você pode utilizar o código abaixo:

```
SELECT 'São Paulo, ' || TO_CHAR(SYSDATE, 'DL') || '.' AS "Data"  
FROM DUAL;
```

Apesar de apresentar uma sintaxe mais simples, o resultado produzido é muito parecido com o anterior. Note as diferenças na figura 16.

Data
1 São Paulo, quarta-feira, 26 de julho de 2017.

**Figura 16:** Data obtida com TO\_CHAR DL.

Agora que você conhece o comando para converter uma data para o formato que deseja, você pode aprender as funções de Data.

## ADD\_MONTHS

A função ADD\_MONTHS retorna a data acrescida dos meses inteiros informados. A sintaxe é a seguinte:

```
ADD_MONTHS(data, inteiro)
```

O argumento data é um valor do tipo date ou outro valor que pode ser convertido para tipo. O argumento inteiro tem que ser um número inteiro. O tipo de retorno é sempre uma data. Se a data for o último dia do mês ou se o mês resultante tiver menos dias, o resultado será o último dia do mês resultante. Caso contrário, o resultado tem o mesmo dia da data. Veja o exemplo do código SQL:

```
SELECT ADD_MONTHS(TO_DATE('310717', 'DDMMYY'), 2) AS "Nova  
data"  
FROM DUAL;
```

A data foi escolhida para exemplificar o que ocorre se o último dia do mês for 31 e no mês subsequente não existir um dia equivalente como

o escolhido. O SGBD indicou que dois meses à frente o último dia do mês é 30 e não 31 como mostra o resultado na figura 17.



**Figura 17:** Resultado do ADD\_MONTHS.

## MONTHS\_BETWEEN

A função *MONTHS\_BETWEEN* retorna o número de meses entre duas datas. A sintaxe da função é:

```
MONTHS_BETWEEN (data1, data2)
```

Se a data1 for posterior à data2, o resultado será positivo. Se data1 for anterior à data2, o resultado será negativo. Se data1 e data2 forem do mesmo mês o resultado será um número inteiro. Caso contrário, será calcula uma parte fracionada com base em um mês de 31 dias. Para experimentar, escreva o comando SQL:

```
SELECT MONTHS_BETWEEN (
    TO_DATE ('310717', 'DDMMYY') ,
    TO_DATE ('310617', 'DDMMYY')
) AS "Meses"
FROM DUAL;
```

O resultado da consulta é 1 mês. A QUERY calcula quantos meses se passaram entre as duas datas.

## NEXT\_DAY

A função *NEXT\_DAY* retorna a data em que se encontra o dia de semana posterior à data informada. A sintaxe é:

```
NEXT_DAY (data, dia_da_semana)
```

O tipo de retorno é sempre do tipo data. O argumento dia\_da\_semana deve ser informado no idioma de sua sessão no SGBD, pode ser completo ou abreviado. Todos os caracteres imediatamente após a abreviatura válida são ignorados. O valor de retorno tem horas, minutos e segundos como qualquer data. Veja o exemplo:

```
SELECT NEXT_DAY( TO_DATE('25052020','DDMMYYYY') , 'Seg' )
FROM DUAL;
```

A QUERY está solicitando qual a próxima segunda-feira após o dia 25/05/2020. A consulta vai retornar o dia 01/06/2020 como sendo a resposta à questão.

## LAST\_DAY

A função *LAST\_DAY* retorna a data do último dia do mês. A sintaxe é apresentada abaixo:

```
LAST_DAY (data1)
```

A função contém um parâmetro data1 e sempre retorna o tipo data. Para ilustrar seu uso, veja o exemplo:

```
SELECT SYSDATE AS "Hoje",
       LAST_DAY(SYSDATE) AS «Último dia do Mês»,
       LAST_DAY(SYSDATE) - SYSDATE AS "Faltam"
  FROM DUAL;
```

O resultado da consulta é apresentado na figura 18.

Hoje	Último dia do mês	Dias que faltam
1 26/07/17	31/07/17	5

**Figura 18:** QUERY com *LAST\_DAY*.

Outro uso pode ser visto no código:

```
SELECT first_name || ' ' || last_name AS "Nome",
       hire_date AS "Data de Contratação",
       LAST_DAY(ADD_MONTHS(hire_date, 3)) AS "Fim da Experiencia"
  FROM EMPLOYEES
 WHERE TO_CHAR(hire_date, 'YYYY')>=2008;
```

A QUERY vai listar o nome completo, a data de contratação, o fim da experiência dos funcionários que foram contratados em 2008 ou mais recente. O fim da experiência ocorre 90 dias após a contratação, no último dia do mês.

## TRUNC

### TRUNC

retorna uma data cortando alguns dados passados por parâmetro. Pode ser feito por dia e mês, utilizando o parâmetro FMT(formato).

A função **TRUNC** (para\_data) retorna a data com uma parte do tempo truncado. A sintaxe é:

```
TRUNC (data1 [, fmt])
```

Você deve informar a data que deseja truncar e pode optar por especificar o formato *fmt*. O valor retornado é sempre do tipo de dados data. Se você omitir *fmt*, a data é truncada para o dia mais próximo. Por exemplo, você pode testar o seguinte SQL:

```
SELECT TO_CHAR(TRUNC(SYSDATE), 'DD/MM/YY HH:MI:SS') AS "COM TRUNC",
       TO_CHAR(SYSDATE, 'DD/MM/YY HH:MI:SS') AS "SEM TRUNC",
  FROM DUAL;
```

Observe o valor que esta *QUERY* retorna na figura 19.

COM TRUNC	SEM TRUNC
26/07/17 12:00:00	26/07/17 06:09:30

**Figura 19:** Comparando TRUNC em Datas.

Os retornos mostram a diferença quando você solicita truncar uma data. Se você não informa nenhum parâmetro simplesmente *TRUNC* a hora, ficando com o dia às 12:00:00. Para testar o comando *TRUNC* com passagem de parâmetros experimente:

```
SELECT TO_CHAR(TRUNC(SYSDATE, 'MM'), 'DD/MM/YY HH:MI:SS') AS "COM TRUNC",
       TO_CHAR(SYSDATE, 'DD/MM/YY HH:MI:SS') AS "SEM TRUNC",
  FROM DUAL;
```

Avalie o valor que esta *QUERY* retorna na figura 20.

COM TRUNC	SEM TRUNC
01/07/17 12:00:00	26/07/17 06:15:53

**Figura 20:** TRUNC com um parâmetro FMT.

Observe que solicitamos que o valor truncasse o mês, assim o mês e o ano foram mantidos, sendo os demais valores cortados e atribuídos os valores iniciais. O dia foi cortado para o dia 01 e a hora para 12:00:00.

## SESSIONTIMEZONE

A função **SESSIONTIMEZONE** retorna o fuso horário da sessão atual. O tipo de retorno é um deslocamento do fuso horário ou o nome da região do fuso horário, dependendo de como o usuário especificou o valor do fuso horário da sessão. Para testar o comando digite:

```
SELECT SESSIONTIMEZONE AS "Nome da Região do Fuso"  
FROM DUAL;
```

### SESSIONTIMEZONE

mostra o fuso horário com base na sessão aberta no banco de dados, mediante sua localização.

Avalie o valor que esta *QUERY* retorna na figura 21.

Nome da Região do Fuso
1 America/Sao_Paulo

**Figura 21:** SESSIONTIMEZONE.

## CURRENT\_DATE

A função **CURRENT\_DATE** retorna a data atual no fuso horário da sessão. O valor de retorno é no calendário gregoriano do tipo data. Para utilizar o comando, experimente:

```
SELECT CURRENT_DATE AS "Data no Fuso"  
FROM DUAL;
```

### CURRENT\_DATE

mostra a data corrente com base na zona de tempo da sessão do usuário.

O resultado da consulta será a data no formato padrão configurado no sistema. Para controlar a consulta, utilize:

```
SELECT TO_CHAR(CURRENT_DATE, 'DD/MM/YYYY HH:MI:SS.SSS') AS  
"Data"  
FROM DUAL;
```

A figura 22 apresenta o resultado da *QUERY*.

Data
1 26/07/17 06:40:09.0909

**Figura 22:** Uso do CURRENT\_DATE.

## FORMATOS DE DATA PARA A SESSÃO

Você pode alterar o formato de sua sessão ou definir um novo formato de data se sua sessão não está no formato correto. Para validar se sua sessão está correta digite:

```
SELECT * FROM V$NLS_PARAMETERS;
```

A consulta deve retornar dados similares aos apresentados na figura 23.

The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view of tables and views under the 'EMPLOYEES' schema. The main area displays a query results grid for the 'v\$nls\_parameters' view. The grid has two columns: 'PARAMETER' and 'VALUE'. The data includes various session parameters like NLS\_LANGUAGE, NLS\_TERRITORY, and NLS\_DATE\_FORMAT, all set to Brazilian Portuguese values. The bottom status bar shows the time as 18:57.

PARAMETER	VALUE
1 NLS_LANGUAGE	BRAZILIAN PORTUGUESE
2 NLS_TERRITORY	BRAZIL
3 NLS_CURRENCY	R\$
4 NLS_ISO_CURRENCY	BRAZIL
5 NLS_NUMERIC_CHARACTERS	,.
6 NLS_CALENDAR	GREGORIAN
7 NLS_DATE_FORMAT	DD/MM/RR
8 NLS_DATE_LANGUAGE	BRAZILIAN PORTUGUESE
9 NLS_CHARACTERSET	AL32UTF8
10 NLS_SORT	WEST_EUROPEAN
11 NLS_TIME_FORMAT	HH24:MI:SSFF
12 NLS_TIMESTAMP_FORMAT	DD/MM/RR HH24:MI:SSFF
13 NLS_TIME_TZ_FORMAT	HH24:MI:SSFF TZR
14 NLS_TIMESTAMP_TZ_FORMAT	DD/MM/RR HH24:MI:SSFF TZR
15 NLS_DUAL_CURRENCY	Cr\$
16 NLS_NCHAR_CHARACTERSET	AL16UTF16
17 NLS_COMP	BINARY
18 NLS_LENGTH_SEMANTICS	BYTE
19 NLS_NCHAR_CONV_EXCP	FALSE

**Figura 23:** Consulta aos parâmetros da sessão.

Você pode alterar alguns desses parâmetros com as seguintes instruções:

```
ALTER SESSION SET NLS_LANGUAGE='BRAZILIAN PORTUGUESE';
ALTER SESSION SET NLS_TERRITORY = 'BRAZIL';
ALTER SESSION SET NLS_NUMERIC_CHARACTERS=',.';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY HH24:MI:SS';
```

## TO\_NUMBER

### TO\_NUMBER

converte uma expressão para um dado tipo NUMBER.

A função **TO\_NUMBER** converte uma expressão para um dado tipo NUMBER. A sintaxe do comando é:

```
TO_NUMBER(expressão [,fmt] [, 'nls parametro'])
```

Onde a expressão pode ser um tipo de dado CHAR, VARCHAR2, NCHAR ou NVARCHAR2 contendo um número no formato especificado ou pode ser um valor BINARY\_FLOAT ou BINARY\_DOUBLE. O formato pode ser especificado de forma opcional no formato *fmt*. Para você compreender, experimente:

```
UPDATE EMPLOYEES
```

```
SET salary = salary + TO_NUMBER('100,00','9G999D00')
WHERE last_name = 'Perkins';
```

Observe que no exemplo a função *TO\_NUMBER* converteu o texto 100,00 em um número. Uma vez feito a conversão, foi efetuado o aumento ao funcionário de sobrenome Perkins. Veja outro exemplo:

```
SELECT TO_NUMBER('3,1415') AS NUMERO FROM DUAL;
```

Veja que a função converteu a expressão texto em um dado do tipo *NUMBER*. Ainda pode-se utilizar para converter moeda. Experimente o código:

```
SELECT TO_NUMBER('$52,502.67', '$99,999.99') FROM DUAL;
```

Como exemplificado, a função permite converter qualquer dado do tipo texto em número. Essa função será muito útil quando você for importar e exportar dados para o SGBD.

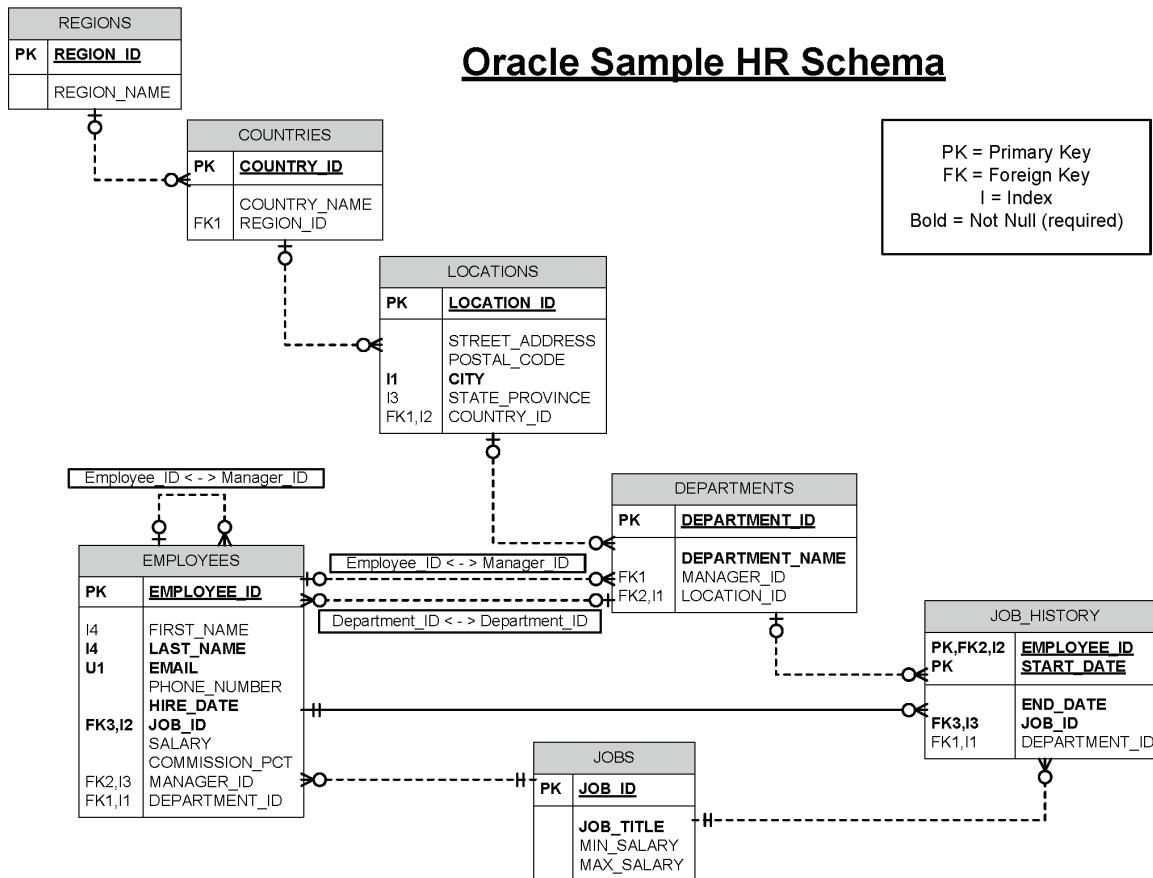


### síntese

Nesta sexta aula, você aprendeu como realizar consultas utilizando *INNER QUERY* e *OUTER QUERY*. Viu como tratar as subconsultas com uma única linha de retorno e como tratar as consultas com múltiplas linhas de retorno. Você também aprendeu a agrupar dados, tratar dados agrupados e como esses dados interagem com as funções de grupo. Seu conhecimento de seleção avançou. Você aprendeu diferentes funções como *AVG*, *SUM*, *COUNT*, *MAX*, *MIN*, *ADD\_MONTHS*, *MONTHS\_BETWEEN*, *NEXT\_DAY*, *LAST\_DAY*, *TRUNC*, *SYSDATE*, *SESSIONTIMEZONE*, *CURRENT\_DATE*, *TO\_CHAR* e *TO\_NUMBER*.

## Atividades

01. Com base no DER, resolva os exercícios:



- a. Mostrar o nome, a data de admissão, cargo e o código do departamento de todos os empregados que tenham o mesmo departamento que o funcionário de nome Steven King. Utilize a tabela EMPLOYEES.

---



---



---



---



---



---



---



---



---



---

- b.** Mostrar o nome, código do departamento e cargo de todos os empregados que trabalhem nos departamentos que fazem parte da cidade 'Seattle'. Utilize as tabelas: EMPLOYEES, DEPARTMENTS e LOCATIONS.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- c.** Exibir código do departamento, nome, cargo de todos os empregados que pertençam ao departamento denominado 'Sales'. Tabelas EMPLOYEES e DEPARTMENTS.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- d.** Selecionar nome, cargo e salário de todos os empregados cujo salário seja maior que a média salarial de todos os empregados.

---

---

---

---

---

- e.** Exibir, nome, salário, data de admissão e código do departamento de todos os empregados que trabalhem no mesmo departamento que o funcionário de nome Steven King e tenham o salário maior que a média salarial dos empregados.

---

---

---

---

---

---

---

---

- f.** Crie uma QUERY que mostre o número do empregado, o último nome e o salário de todos os empregados que ganhem mais que a média salarial. Ordenar o resultado em ordem crescente de salários. Coloque um alias no cabeçalho da cada coluna.

---

---

---

---

---

---

---

- g.** Crie uma QUERY que mostre o último nome, o número do departamento e o código do cargo de todos os empregados cuja localização do departamento seja igual a 1700

---

---

---

---

---

---

---

- h.** Crie uma QUERY que mostre o último nome e o salário de todos os funcionários que tenham o King como gerente.

---

---

---

---

---

---

---

- i.** Crie uma QUERY que mostre o número do departamento, o último nome e o código do cargo para cada empregado que esteja no departamento 'Executive'

---

---

---

---

---

---

---

## Anotações



# AULA 8

## FUNÇÕES EM SQL

### NESTA AULA

- » Introdução a Funções SQL
- » Função para valores nulos - NVL
- » Funções alfanuméricas
- » Funções numéricas

### ■ METAS DE COMPREENSÃO

- » Conhecer as principais funções do SQL.
- » Aprender a utilizar as funções para resolver problemas.
- » Fixar o conhecimento do SQL lidando com diferentes comandos.

### ■ APRESENTAÇÃO

Nesta aula você irá aprender a utilizar as funções no SQL. Fixar os conceitos vistos até esta aula vai lhe ajudar a elaborar consultas complexas. A cada nova instrução você vai rever e utilizar os conceitos do comando SELECT, avançando no domínio da SQL.

Para que você possa entender estes conceitos, apresentaremos as funções no SQL, bem como as funções que o SGBD oferece para tratar, importar e exibir dados.

# ■ INTRODUÇÃO A FUNÇÕES SQL

Nesta aula você vai aprender sobre as principais funções utilizadas para manipular dados. A maioria das funções aceitam um ou mais argumentos e retorna um valor. Um argumento pode ser uma constante, pode se referir a uma variável ou uma coluna. A sintaxe para grande parte das funções é a seguinte:

```
FUNÇÃO_NOME (argumento1, argumento2, ...)
```

As funções podem ser usadas para realizar cálculos sobre datas, modificar valores de itens individuais, manipular saída de dados para grupos de linhas e alterar formatos de datas para mostrá-los.

## ■ FUNÇÃO PARA VALORES NULOS - NVL

É uma função que retorna um valor em um campo nulo.

Um valor nulo é um valor indisponível e desconhecido. O valor nulo não é zero, zero é um número enquanto que nulo é ausência de valor. Se algum valor de uma coluna em uma expressão for nulo, o resultado será nulo. Para exemplificar experimente digitar a declaração SQL.

```
SELECT      first_name || ' ' || last_name AS "Nome",
            salary * 12 + commission_pct AS "Salário Anual"
  FROM    EMPLOYEES
 WHERE      first_name LIKE 'A%';
```

Observe o resultado que aparece na remuneração na coluna Salário Anual dos funcionários que tem A no início de seus nomes. Alguns dados não serão calculados pois a coluna commission\_pct armazena a comissão dos vendedores, mantendo valores nulos para os demais funcionários. Como a coluna de comissão apresenta valor nulo, todo o cálculo se torna proibitivo. O resultado da consulta é apresentado na figura 01.

	Nome	Salário Anual
1	Amit Banda	74400,1
2	Alexis Bull	(null)
3	Anthony Cabrio	(null)
4	Alberto Errazuriz	144000,3
5	Adam Fripp	(null)
6	Alexander Hunold	(null)
7	Alyssa Hutton	105600,25
8	Alexander Khoo	(null)
9	Allan McEwen	108000,35
10	Alana Walsh	(null)

**Figura 01:** Consulta sem NVL.

Contudo, você deseja que o resultado do salário anual dos funcionários sem comissão seja calculado e exibido junto com os vendedores. Para corrigir a questão, você deve utilizar a função NVL que converte valores nulos para não nulos. A função NVL conta com dois argumentos. Observe a sintaxe:

`NVL (expressão, valor)`

Onde:

Expressão pode ser uma coluna do tipo data, uma coluna numérica ou uma *string*.

Valor será o valor que irá substituir o *NULL* que a *QUERY* irá retornar.

Para realizar o cálculo para todos os empregados, é necessário converter os valores nulos para numéricos. Experimente o código SQL:

```
SELECT      first_name || ' ' || last_name AS "Nome",
salary * 12 + NVL(commission_pct,0) AS "Salário Anual"
FROM    EMPLOYEES
WHERE first_name LIKE 'A%';
```

O resultado é apresentado na figura 02.

	Nome	Salário Anual
1	Amit Banda	74400,1
2	Alexis Bull	49200
3	Anthony Cabrio	36000
4	Alberto Errazuriz	144000,3
5	Adam Fripp	98400
6	Alexander Hunold	108000
7	Alyssa Hutton	105600,25
8	Alexander Khoo	37200
9	Allan McEwen	108000,35
10	Alana Walsh	37200

**Figura 02:** QUERY com NVL.

Observe que com o uso da função *NVL*, você corrige a questão dos salários nulos, contudo falta formatar o salário. Utilize:

```
SELECT      first_name || ' ' || last_name AS "Nome",
            TO_CHAR( salary * 12 + NVL(commission_pct,0),
            'FML099G999D00') AS "Salário Anual"
FROM    EMPLOYEES
WHERE      first_name LIKE 'A%';
```

A figura 03 apresenta como fica o retorno da *QUERY* após receber o tratamento de dados com *TO\_CHAR* e *NVL*.

	Nome	Salário Anual
1	Amit Banda	R\$074.400,10
2	Alexis Bull	R\$049.200,00
3	Anthony Cabrio	R\$036.000,00
4	Alberto Errazuriz	R\$144.000,30
5	Adam Fripp	R\$098.400,00
6	Alexander Hunold	R\$108.000,00
7	Alyssa Hutton	R\$105.600,25
8	Alexander Khoo	R\$037.200,00
9	Allan McEwen	R\$108.000,35
10	Alana Walsh	R\$037.200,00

**A figura 03:** QUERY com *NVL* e *TO\_CHAR*.

## ■ FUNÇÕES ALFANUMÉRICAS

Agora você vai conhecer as principais funções alfanuméricas. As funções alfanuméricas aceitam dados alfanuméricos e podem retornar alfanumérico ou valores numéricos.

### LOWER

#### LOWER

função que retorna string com todas as letras minúsculas.

A função **LOWER** retorna uma *string* com todas as letras minúsculas. A sintaxe é:

**LOWER** (*expressão*)

Onde:

Expressão pode ser qualquer dos tipos de dados *CHAR*, *VARCHAR2*, *NCHAR*, *NVARCHAR2*, *CLOB* ou *NCLOB*. O valor de retorno é do mesmo tipo de dados. O banco de dados define o caso dos caracteres com base no mapeamento binário definido para o conjunto de caracteres subjacente. Para mostrar o nome dos departamentos e a constante *BANCO DE DADOS* em letra minúscula, faça:

```
SELECT      LOWER(department_name) AS "Nome do Departamento",
            LOWER('BANCO DE DADOS') AS "Constante"
FROM    DEPARTMENTS
ORDER BY 1;
```

O resultado da *QUERY* é apresentado na figura 04. Note que todas as linhas retornadas estão com valores em minúsculas.

Nome do Departamento	Constante
1 accounting	banco de dados
2 administration	banco de dados
3 benefits	banco de dados
4 construction	banco de dados
5 contracting	banco de dados
6 control and credit	banco de dados
7 corporate tax	banco de dados
8 executive	banco de dados
9 finance	banco de dados
10 government sales	banco de dados
11 human resources	banco de dados
12 it	banco de dados
13 it helpdesk	banco de dados
14 it support	banco de dados
...	

**Figura 04:** Função *LOWER* na *QUERY*.

Você também pode utilizar a função junto às cláusulas *WHERE*. Observe a *QUERY*:

```
SELECT      first_name AS nome, job_id AS cargo, email
FROM    EMPLOYEES
WHERE      LOWER(first_name) = 'steven';
```

A figura 05 apresenta o retorno de dados da *QUERY*.

	NOME	CARGO	EMAIL
1	Steven	AD_PRES	SKING
2	Steven	ST_CLERK	SMARKLE

**Figura 05:** LOWER junto ao WHERE.

Note que foi utilizada a função *LOWER* na *WHERE*, ou seja, condicionando a comparação. O SGBD converte todos os nomes e compara com a *string* *steven* em minúsculas. As linhas que satisfazem a condição são então exibidas como foram armazenadas no SGBD.

## UPPER

### UPPER

função que retorna string com todas as letras maiúsculas.

A função **UPPER** retorna uma *string* com todas as letras maiúsculas.

A sintaxe é:

`UPPER (expressão)`

Onde:

Expressão pode ser qualquer dos tipos de dados *CHAR*, *VARCHAR2*, *NCHAR*, *NVARCHAR2*, *CLOB* ou *NCLOB*. O valor de retorno é do mesmo tipo de dados. O banco de dados define o caso dos caracteres com base no mapeamento binário definido para o conjunto de caracteres subjacentes. Para mostrar o nome dos Funcionários em maiúsculo, faça:

```
SELECT UPPER(first_name) || ' ' || UPPER(last_name) AS NOME
FROM EMPLOYEES
ORDER BY 1;
```

Você pode visualizar o resultado da *QUERY* na figura 06. Observe que todos os nomes nas linhas estão com maiúsculas.

	NOME
1	ADAM FRIPP
2	ALANA WALSH
3	ALBERTO ERRAZURIZ
4	ALEXANDER HUNOLD
5	ALEXANDER KHOO
6	ALEXIS BULL
7	ALLAN MCEWEN
8	ALYSSA HUTTON
9	AMIT BANDA
10	ANTHONY CABRIO
11	BRITNEY EVERETT
	...

**Figura 06:** QUERY com UPPER.

Igual a função *LOWER*, a função *UPPER* pode ser utilizada junto às cláusulas *WHERE*. Observe a *QUERY*:

```
SELECT      first_name AS nome, job_id AS cargo, email  
FROM    EMPLOYEES  
WHERE  UPPER(first_name) = 'DAVID';
```

A figura 07 apresenta o retorno de dados da *QUERY*.

	NOME	CARGO	EMAIL
1	David	IT_PROG	DAUSTIN
2	David	SA REP	DBERNSTE
3	David	SA REP	DLEE

**Figura 07:** *UPPER* junto ao *WHERE*.

Foi utilizada a função *UPPER* condicionando a comparação na *WHERE*. A exibição dos dados permanece como cadastrado, mas a escolha das tuplas se dá mediante a conversão de todos os nomes para maiúsculas e comparação com a string DAVID.

## INITCAP

A função **INITCAP** retorna uma *string* com a primeira letra de cada palavra em maiúscula, todas as outras letras em minúsculas. As palavras são delimitadas por espaços em branco ou caracteres que não são alfanuméricos. A sintaxe é:

```
INITCAP (expressão)
```

### INITCAP

função que retorna string com a primeira letra de cada palavra em maiúscula e todas as outras letras em minúsculas.

Onde:

Expressão pode ser qualquer dos tipos de dados *CHAR*, *VARCHAR2*, *NCHAR* ou *NVARCHAR2*. O valor de retorno é do mesmo tipo de dados. O banco de dados define o caso dos caracteres iniciais com base no mapeamento binário definido para o conjunto de caracteres subjacentes. É importante destacar que esta função não suporta dados *CLOB* diretamente. No entanto, os dados *CLOB* podem ser transmitidos como argumentos por meio da conversão de dados implícita.

```

SELECT 'Funcionário: '||INITCAP(first_name)||' '||last_name
Nome,
FROM EMPLOYEES
WHERE last_name LIKE '%g'
ORDER BY 1;

```

O resultado pode ser avaliado na figura 08 onde você poderá notar que os nomes estão todos grafados com a primeira letra de cada palavra em maiúscula e todas as demais são minúsculas.

NOME
1 Funcionário: Janette King
2 Funcionário: Kelly Chung
3 Funcionário: Nancy Greenberg
4 Funcionário: Payam Kaufling
5 Funcionário: Renske Ladwig
6 Funcionário: Steven King

**Figura 08:** Resultado da QUERY com INITCAP.

A função *INITCAP* pode ser utilizada junto a cláusula *WHERE*. Observe a *QUERY*:

```

SELECT      first_name AS nome, phone_number AS telefone
FROM        EMPLOYEES
WHERE       INITCAP(first_name) = 'Kelly';

```

A figura 09 apresenta o retorno de dados da *QUERY*.

NOME	TELEFONE
1 Kelly	650.505.1876

**Figura 09:** Retorno da QUERY com INITCAP na WHERE.

## RPAD

**RPAD**  
função que completa a direita de uma coluna com caracteres especificados. A sintaxe da função é:

RPAD (expressão\_1, n, expressão\_1)

Onde:

Expressão\_1 é a coluna ou *string* enviada ou original.  
N é o comprimento de caracteres.  
Expressão\_2 é o caractere que será replicado quantas vezes for necessário.

Esta função é útil para formatar a saída de uma consulta. Tanto expressão\_1 como expressão\_2 podem ser quaisquer dos tipos de dados CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB ou NCLOB. A sequência de caracteres que retorna é de tipo VARCHAR2. A string de retorno será do mesmo tipo de dados que a expressão\_1. O argumento n deve ser um NÚMERO inteiro ou um valor que pode ser convertido em um NÚMERO inteiro. Para experimentar, digite o código:

```
SELECT RPAD(department_name, 30, '_') AS "Nome do Departamento"  
FROM DEPARTMENTS;
```

Você pode visualizar o resultado da QUERY na figura 10.

	Nome do Departamento
1	Administration_____
2	Marketing_____
3	Purchasing_____
4	Human Resources_____
5	Shipping_____
6	IT_____
7	Public Relations_____
8	Sales_____
9	Executive_____
10	Finance_____
11	Accounting_____
	...

**Figura 10:** RPAD.

Como você pode observar, o RPAD completou no lado *RIGHT* ou na direita com o caractere '\_' até completar 30 caracteres. Vale ressaltar que a expressão\_1 não pode ser nula. Se você não especificar a expressão\_2, então será definido como padrão um único espaço em branco. Caso a expressão\_1 for maior do que n, então a função RPAD retorna apenas a parte que se encaixa em n. Lembre-se que n é o comprimento total que se espera de retorno. Veja outro exemplo:

```

SELECT      first_name||' '|| last_name AS NOME,
            TO_CHAR(salary,'fm19G999D00') AS SALARIO,
            RPAD(' ',salary/1000,'*') AS GRAFICO
FROM    EMPLOYEES
WHERE      department_id = 50
        AND salary BETWEEN 3800 AND 10000
ORDER BY 1;

```

A figura 11 apresenta o resultado do código SQL. Observe que você está criando um modelo gráfico para representar o salário do funcionário.

	NOME	SALARIO	GRAFICO
1	Adam Fripp	R\$8.200,00	*****
2	Alexis Bull	R\$4.100,00	***
3	Britney Everett	R\$3.900,00	**
4	Kelly Chung	R\$3.800,00	**
5	Kevin Mourgos	R\$5.800,00	****
6	Matthew Weiss	R\$8.000,00	*****
7	Nandita Sarchand	R\$4.200,00	***
8	Payam Kaufling	R\$7.900,00	*****
9	Sarah Bell	R\$4.000,00	***
10	Shanta Vollman	R\$6.500,00	*****

**Figura 11:** RPAD como saída gráfica.

O resultado da QUERY anterior utilizou o argumento n como sendo uma saída de tamanho variado. O n foi calculado pela divisão do salário do empregado por 1000. Como o RPAD só utiliza inteiro, os valores fracionados da conta são descartados. A coluna GRÁFICO recebe na expressão\_1 o valor padrão e a expressão\_2 recebe o caractere \*. Note que é possível ir além com a função RPAD do que apenas preparar os dados para relatórios, contudo, será raro a utilização para outros fins.

## LPAD

### LPAD

função que completa a esquerda de uma coluna com caracteres especificados. A sintaxe da função é:

**LPAD** (expressão\_1, n, expressão\_1)

Onde:

Expressão\_1 é a coluna ou *string* enviada ou original.  
N é o comprimento de caracteres.  
Expressão\_2 é o caractere que será replicado quantas vezes for necessário.

Esta função é útil para formatar a saída de uma consulta. Expressão\_1 e expressão\_2 podem ser *CHAR*, *VARCHAR2*, *NCHAR*, *NVARCHAR2*, *CLOB* ou *NCLOB*. A sequência de caracteres que retorna é de tipo *VARCHAR2*. A *string* de retorno será do mesmo tipo de dados que a expressão\_1. O argumento n deve ser um número inteiro ou um valor que pode ser convertido em um número inteiro. Digite o código e avalie o exemplo:

```
SELECT LPAD(department_id,20,'*'),
       LPAD(department_id,40),
       LPAD(department_id,10,'.')
FROM DEPARTMENTS
WHERE department_id IN (10,20,30);
```

A figura 12 apresenta o retorno da *QUERY*.

	LPAD(DEPARTMENT_ID,20,'*')	LPAD(DEPARTMENT_ID,40)	LPAD(DEPARTMENT_ID,10,'.')
1	*****10		10 .....
2	*****20		20 .....
3	*****30		30 .....

**Figura 12:** Formas de usar *LPAD*.

A forma mais comum de uso do *LPAD* é junto com *RPAD*, um tratando o lado direito e o outro comando tratando o lado esquerdo. O exemplo a seguir mostra o uso de ambos os comandos em uma *QUERY*.

```
SELECT      RPAD(department_name,30,'_') ||
            LPAD(department_id,5,'_')
            AS "Nome do Departamento e Código"
FROM        DEPARTMENTS
WHERE       department_id<110;
```

A *QUERY* irá exibir o nome e o código dos departamentos em uma única coluna. Deve-se exibir apenas os departamentos que tem código menor que 110. A figura 13 exibe o resultado da *QUERY*.

Nome do Departamento e Código	
1 Administration	10
2 Marketing	20
3 Purchasing	30
4 Human Resources	40
5 Shipping	50
6 IT	60
7 Public Relations	70
8 Sales	80
9 Executive	90
10 Finance	100

**Figura 13:** RPAD e LPAD.

Este é o caso mais comum de utilização das funções *RPAD* e *LPAD*, tratando os dados para criar um relatório organizado.

## SUBSTR

### SUBSTR

função que retorna uma parte de uma string.

`SUBSTR (expressão, posição[, quantia])`

Onde:

Expressão é a coluna ou *string* enviada ou original.

Posição é a posição inicial para retirar parte da *string*.

Quantia é a quantidade de caracteres a partir da posição inicial.

Para exemplificar, analise o código:

```
SELECT      first_name || ' ' || last_name AS Nome,
            SUBSTR(first_name||last_name, 3, 4) ||
            TO_CHAR(hire_date, 'MMDD') ||
            AS Senha
FROM    EMPLOYEES;
```

Observe o resultado da consulta na figura 14.

	NOME	senha
1	Steven King	even0617
2	Neena Kochhar	enaK0921
3	Lex De Haan	xDe 0113
4	Alexander Hunold	exan0103
5	Bruce Ernst	uceE0521
6	David Austin	vidA0625
7	Valli Pataballa	lliP0205
8	Diana Lorentz	anaL0207
9	Nancy Greenberg	ncyG0817
10	Daniel Faviet	niel0816
...		

**Figura 14:** QUERY com SUBSTR.

A instrução *SUBSTR* irá recortar um pedaço da *string*, o corte inicia na 3<sup>a</sup> posição e a partir daí avança 4 caracteres, ou seja, vai até a posição 7.

## INSTR

A função **INSTR** retorna a posição de ocorrência de um caractere ou *substring*. A função retorna um número inteiro que indica onde foi encontrado o caractere dentro da *string*. A sintaxe da função é:

```
INSTR (expressão, o_que_procura[, posição][, ocorrência])
```

### INSTR

função que retorna a posição de ocorrência de um caractere ou *substring*.

Onde:

Expressão é a coluna ou *string* enviada ou original.

O\_que\_procura uma *substring* de caracteres

Posição é a posição inicial para procurar, é um número inteiro diferente de zero. Se a posição for negativa, o SGBD conta de trás para o início da *string* e, em seguida, procura novamente para trás a partir da posição resultante.

Ocorrência é um número inteiro, positivo que indica a quantidade de ocorrências do caractere.

Se a pesquisa não tiver êxito, não for possível encontrar a *substring*, não existir ocorrência, então o valor de retorno será 0. Para testar o comando use:

```
SELECT INSTR('Professor Claudiney Sanches Júnior','in') AS  
"Posição"  
FROM DUAL;
```

A consulta irá retornar a posição 16. Observe que o primeiro parâmetro foi a *string* e a segunda foi a sequência que você deseja encontrar. A *QUERY* retorna a posição que encontrou a primeira ocorrência. Veja como fica o comando com o parâmetro posição:

```
SELECT INSTR('DBA ganha bem!', 'a', 7) AS "Posição"  
FROM DUAL;
```

Esta consulta passa três argumentos, sendo de interesse o parâmetro posição. Ao informar que a posição é 7, você está solicitando que busque a ocorrência do 'a' após a posição 7. Assim a *QUERY* vai retornar 9 identificando a letra 'a' presente no final da palavra ganha. Para finalizar a instrução, experimente:

```
SELECT INSTR('Banco de Dados é muito legal!', 'a', 1,3) AS  
"Posição"  
FROM DUAL;
```

Você vai obter o valor 27 como retorno da consulta, pois neste caso, você está solicitando que a função retorne a posição da terceira ocorrência do caractere 'a'. A pesquisa inicia na posição 1 e deve percorrer até encontrar a terceira ocorrência.

## LENGTH

### LENGTH

função que retorna a quantidade de caracteres que uma expressão ou campo tem.

A função **LENGTH** retorna a quantidade de caracteres que uma expressão ou campo tem. A sintaxe é:

**LENGTH** (*expressão*)

Onde:

Expressão é a coluna ou *string* enviada ou original.

Para exemplificar, analise o código:

```
SELECT department_name AS "Nome do Departamento",
       LENGTH(department_name) AS "Tamanho do Nome"
  FROM DEPARTMENTS;
```

Observe na figura 15 o uso que foi dado para a função *LENGTH* na *QUERY*. A função conta a quantidade de caracteres presente em cada nome de departamento. A função *LENGTH* é útil quando você trata dados nos processos de importação e exportação de base.

Nome Departamento	Tamanho do nome
1 Administration	14
2 Marketing	9
3 Purchasing	10
4 Human Resources	15
5 Shipping	8
6 IT	2
7 Public Relations	16
8 Sales	5
9 Executive	9
10 Finance	7
11 Accounting	10
12 Treasury	8
13 Corporate Tax	13
14 Control And Credit	18
15 Shareholder Services	20
16 Benefits	8
17 Manufacturing	13
18 Construction	12
19 Contracting	11
20 Operations	10
21 IT Support	10
22 NOC	3
...	

**Figura 15:** QUERY com *LENGTH*.

Você também pode utilizar a variação do comando para ter outros tamanhos. Um bem interessante é saber quantos *bytes* uma determinada *string* consome. Para isso utilize **LENGTHB** como exemplificado no SQL:

```
SELECT LENGTHB('Escreva aqui o que quer saber') AS "Tamanho
em bytes"
  FROM DUAL;
```

#### LENGTHB

função que retorna a quantidade de bytes de uma expressão ou campo.

A função retorna o valor apresentado na figura 16.

Tamanho em bytes	
1	
	25

**Figura 16:** QUERY com LENGTHB.

A QUERY retornou quantos bytes são necessários para armazenar a expressão informada. Isso é especialmente útil quando os dados migram para dispositivos com limitações como por exemplo a plataforma *mobile*.

## TRANSLATE

### TRANSLATE

função que substitui um caractere em uma string.

A função **TRANSLATE** substitui um caractere em uma *string*. A sintaxe é:

```
TRANSLATE (expressão, de_valor, para_valor)
```

Onde:

Expressão é a coluna ou *string* enviada ou original.

de\_valor é o caractere que irá ser localizado.

para\_valor é o caractere que será sobreposto onde de\_valor for localizado.

*TRANSLATE* retorna a expressão com todas as ocorrências de\_valor substituídas pelo caractere correspondente em para\_valor. Os caracteres na expressão que não correspondem a de\_valor não são substituídos. A expressão que é uma *string* deve ser colocada entre aspas simples. O argumento para\_valor deve ser um caractere. *TRANSLATE* permite que você faça várias substituições em uma única operação. Para experimentar o uso da função, digite:

```
SELECT first_name AS "Primeiro Nome", TRANSLATE(UPPER(first_name), 'L', 'X') AS "Usuário"  
FROM EMPLOYEES;
```

A figura 17 apresenta o resultado da consulta.

	Primeiro Nome	Usuário
1	Ellen	EXXEN
2	Sundar	SUNDAR
3	Mozhe	MOZHE
4	David	DAVID
5	Hermann	HERMANN
6	Shelli	SHEXXI
7	Amit	AMIT
8	Elizabeth	EXIZABETH
9	Sarah	SARAH
10	David	DAVID
11	Laura	XAURA
12	Harrison	HARRISON

**Figura 17:** TRANSLATE.

Observe que a função percorre todos os registros e substitui o caractere L por X criando a coluna de sugestão de *logon* ou nome dos usuários. Um segundo exemplo irá utilizar o comando para fazer múltiplas substituições. Experimente:

```
SELECT TRANSLATE('Você será um DBA', 'ê/á/ ', 'e/a/_')
AS "Função TRANSLATE"
FROM DUAL;
```

O resultado é apresentado na figura 18.

Função TRANSLATE
1 Voce_sera_um_DBA

Figura 18. TRANSLATE com múltiplas substituições.

Observe que neste exemplo a função substituiu os espaços em branco pelo '\_' e os caracteres acentuados por caracteres normais em uma única instrução.

## REPLACE

A função **REPLACE** é muito parecida com a função *TRANSLATE*, a diferença é que a função *TRANSLATE* substitui um caractere por outro enquanto a função **REPLACE** substitui uma expressão por outra expressão. A sintaxe é apresentada a seguir:

### REPLACE

função que substitui uma expressão por outra expressão.

```
REPLACE (expressão, de_valor [,para_valor])
```

Onde:

Expressão é a coluna ou *string* enviada ou original.  
de\_valor é a *string* que irá ser localizada.  
para\_valor é a *string* que será sobreposta onde de\_valor for localizada.

A função *REPLACE* retorna uma *string* com todas as ocorrências de\_valor substituídas por para\_valor. Se para\_valor for omitido ou nulo, todas as ocorrências de de\_valor serão removidas. Veja o código de exemplo:

```
SELECT job_title,  
REPLACE(job_title,'Finance Manager','Gerente Financeiro') AS  
"Cargo"  
FROM JOBS  
WHERE job_id = 'FI_MGR';
```

A figura 19 apresenta o resultado da consulta.

JOB_TITLE	CARGO
Finance Manager	Gerente Financeiro

**Figura 19:** *REPLACE*.

Nesta consulta o título do cargo foi substituído de 'Finance Manager' para 'Gerente Financeiro'. Outra substituição que o *REPLACE* faz é de trocar parte da *string* por outra parte. Veja o próximo exemplo:

```
SELECT job_title,  
REPLACE(job_title,'nt','xxxx') CARGO  
FROM JOBS;
```

O resultado da consulta é apresentado na figura 20.

JOB_TITLE	CARGO
1 President	PresideXXXX
2 Administration Vice President	Administration Vice PresideXXXX
3 Administration Assistant	Administration AssistaXXXX
4 Finance Manager	Finance Manager
5 Accountant	AccouXXXXaXXXX
6 Accounting Manager	AccouXXXXing Manager
7 Public Accountant	Public AccouXXXXaXXXX
8 Sales Manager	Sales Manager
9 Sales Representative	Sales RepreseXXXXtative
10 Purchasing Manager	Purchasing Manager
11 Purchasing Clerk	Purchasing Clerk
12 Stock Manager	Stock Manager

...

**Figura 20:** REPLACE substituindo parte da string.

No exemplo acima, a QUERY procurou a ocorrência da sequência de caracteres 'nt' e substituiu por 'XXXX'. Para finalizar, você pode omitir o argumento para\_valor e neste caso a função REPLACE irá remover o argumento de\_valor da expressão. Experimente o seguinte código SQL:

```
SELECT job_title,
REPLACE(job_title, ' Manager') CARGO
FROM JOBS;
```

O resultado da QUERY é exibido na figura 21.

JOB_TITLE	CARGO
1 President	President
2 Administration Vice President	Administration Vice President
3 Administration Assistant	Administration Assistant
4 Finance Manager	Finance
5 Accountant	Accountant
6 Accounting Manager	Accounting
7 Public Accountant	Public Accountant
8 Sales Manager	Sales
9 Sales Representative	Sales Representative
10 Purchasing Manager	Purchasing
11 Purchasing Clerk	Purchasing Clerk
12 Stock Manager	Stock

**Figura 21:** REPLACE removendo caracteres da string.

Observe que foi removido a sequência de caracteres ' Manager' de todos os cargos. Assim esta função é muito interessante ao tratar dados em migrações de SGBD, importar e exportar dados.

# FUNÇÕES NUMÉRICAS

## ROUND

### ROUND

função que arredonda o valor.

A função **ROUND** arredonda o valor. A sintaxe é:

```
ROUND (numero_valor [, inteiro])
```

Onde:

numero\_valor é o número que irá ser arredondado.

inteiro é número de casas decimais, se for omitido será atribuído 0 e não terá casas decimais.

O argumento inteiro pode ser negativo. Neste caso irá arredondar os dígitos à esquerda do ponto decimal. Numero\_valor pode ser qualquer tipo de dado numérico ou qualquer tipo de dado não-numérico que possa ser implicitamente convertido em um tipo de dado numérico. O argumento inteiro deve ser um número inteiro. Se você omitir o número inteiro, a função retorna o mesmo tipo de dado do tipo numero\_valor. Se você incluir inteiro, a função retornará um dado do tipo *NUMBER*. Consulte os exemplos para entender como funciona a função de arredondamento.

```
SELECT ROUND(45.193,1) "Round" FROM DUAL;
```

A QUERY retorna 45.2 como demonstrado na figura 22.

Round	
1	45,2

**Figura 22:** ROUND(15.193,1).

```
SELECT ROUND(45.86,1) "Round" FROM DUAL;
```

A QUERY retorna 45.9 como demonstrado na figura 23. O arredondamento subiu devido à parte fracionada 0.86 como queríamos uma casa após o ponto decimal a função analisou o próximo número 45.X6 onde

X é o número que pode sofrer arredondamento e 6, o próximo número depois do determinante. Como 6 é maior ou igual a 5 então X será arredondado para cima, deve ser acrescentado em um.

Round	
1	45,9

**Figura 23:** ROUND(45.86 ,1).

Veja como seria o mesmo arredondamento caso o número em questão fosse 45.84.

```
SELECT ROUND(45.84,1) "Round" FROM DUAL;
```

A QUERY retorna 45.8 pois indicamos que queremos uma casa decimal, assim o 8 poderá sofrer arredondamento. Para avaliar o próximo número é analisado, sendo ele 4 não é maior ou igual a 5, assim não será incrementado o 8. Veja o retorno da QUERY na figura 24.

Round	
1	45,8

**Figura 24:** ROUND(45.84, 1).

Outro exemplo pode ser visto na consulta abaixo:

```
SELECT ROUND(45.84) "Round" FROM DUAL;
```

A QUERY retorna 46 visto que você não passou o argumento inteiro como parâmetro. Neste caso a função entende que o argumento inteiro é 0 e retorna apenas o número inteiro que não terá casas decimais. O número de retorno sofrerá o arredondamento, visto que estamos avaliando 4X como o número base. A próxima casa ou o próximo número é o número 8 que é maior ou igual a 5. Assim o número X, ou seja, neste momento 5 deve ser incrementado. Veja na figura 25 o resultado da consulta.

Round	
1	46

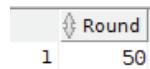
**Figura 25:** ROUND(45.84).

O caso mais complexo é o arredondamento com passagem de parâmetros negativos. Neste caso você está solicitando que seja arredon-

dado a dezena, centena, milhar. Para entender melhor teste a *QUERY*:

```
SELECT ROUND(45.84,-1) "Round" FROM DUAL;
```

Para entender melhor, imagine que a *QUERY* desloca a casa decimal temporariamente. Assim você tem  $ROUND(4.584, 0)$ . Neste caso, 4 será avaliado para arredondamento, e o número que irá determinar se o arredondamento será para cima ou para baixo é o próximo número que é 5. Como 5 é maior ou igual fica determinado que o arredondamento será para cima. Então o 4 é incrementado e se torna 5.0 como a operação tinha deslocado a casa decimal para a esquerda, para finalizar ela deve-se voltar a casa decimal para sua posição. Assim o resultado é 50 como demonstra a figura 26.



**Figura 26:**  $ROUND(45.84, -1)$ .

É importante salientar que você pode fazer cálculos e solicitar simultaneamente que a função *ROUND* faça o arredondamento. Por exemplo, imagine que você queira listar o salário de todos os funcionários acrescidos do bônus de férias de 1/3 previsto para CLT no Brasil. Você pode utilizar o seguinte código SQL:

```
SELECT first_name || ' ' || last_name NOME,  
TO_CHAR(salary, 'FML09G999D00') SALARIO,  
TO_CHAR(ROUND(salary*1.30,2), 'FML09G999D00') "Salário de Férias"  
FROM EMPLOYEES;
```

Observe que a consulta solicita o nome completo do funcionário, o atual salário, sendo que o valor deve estar corretamente formatado e o valor de remuneração acrescido de 1/3 da remuneração das férias. Para facilitar a construção da *QUERY* o cálculo foi sintetizado em um fator, ou seja, poderia ser escrito como:  $\text{salário} * 30 / 100 + \text{salário}$ . Este cálculo é o mesmo que  $\text{salario} * 1.3$  só que neste formato as operações foram simplificadas e estão mais otimizadas a nível computacional. A figura 27 apresenta o retorno da *QUERY*

	NOME	SALARIO	Salário de Férias
1	Steven King	R\$24.000,00	R\$31.200,00
2	Neena Kochhar	R\$17.000,00	R\$22.100,00
3	Lex De Haan	R\$17.000,00	R\$22.100,00
4	Alexander Hunold	R\$09.000,00	R\$11.700,00
5	Bruce Ernst	R\$06.000,00	R\$07.800,00
6	David Austin	R\$04.800,00	R\$06.240,00
7	Valli Pataballa	R\$04.800,00	R\$06.240,00
8	Diana Lorentz	R\$04.200,00	R\$05.460,00
9	Nancy Greenberg	R\$12.008,00	R\$15.610,40
10	Daniel Faviet	R\$09.000,00	R\$11.700,00
11	John Chen	R\$08.200,00	R\$10.660,00
12	Ismael Sciarra	R\$07.700,00	R\$10.010,00
...			

**Figura 27:** Retorno da QUERY do Salário de Férias.

## TRUNC

Você já viu a função **TRUNC** trabalhando com data, mas agora você verá a mesma função lidando com números. A função corta o número de acordo com o solicitado. Sua sintaxe é:

`TRUNC (n1 [, n2])`

Onde:

n1 é o valor que será cortado ou truncado.

n2 é o número de casas decimais. Se n2 for omitido, então n1 é truncado para 0 lugares. N2 pode ser negativo para fazer zeros dígitos antes do ponto decimal.

Esta função toma como argumento qualquer tipo de dado numérico ou qualquer tipo de dado não-numérico que possa ser convertido em um tipo de dado numérico. Se você omitir n2, a função retornará o mesmo tipo de dados com o tipo de dados numérico do argumento. Se você incluir n2, a função retornará *NUMBER*. Analise os exemplos para entender como funciona a função *TRUNC*.

```
SELECT TRUNC(45.193,1) "Trunc" FROM DUAL;
```

A QUERY retorna 45.1 como demonstrado na figura 28.

### TRUNC

função que quando recebe um número como argumento, corta o número de acordo com o solicitado.

		Trunc
1		45,1

**Figura 28:** TRUNC(15.193,1).

```
SELECT TRUNC(45.86,1) "Trunc" FROM DUAL;
```

A QUERY retorna 45.8 como demonstrado na figura 29. Isso ocorre porque não há arredondamento e sim um corte no número, mantendo apenas um número após o ponto decimal.

		Trunc
1		45,8

**Figura 29:** TRUNC(45.86,1).

Veja como seria se você solicitar truncar o número 45.84.

```
SELECT TRUNC(45.84,1) "Trunc" FROM DUAL;
```

A QUERY retorna 45.8 pois você indicou que quer uma casa decimal, assim o corte ocorre no número 8. O retorno da QUERY continua sendo igual ao apresentado na figura 29. Outro exemplo pode ser visto na consulta abaixo:

```
SELECT TRUNC(45.84) "Trunc" FROM DUAL;
```

A QUERY retorna 45 visto que você não passou o argumento n2 como parâmetro. Neste caso a função entende que o argumento n2 é 0 e retorna apenas o número inteiro sem casas decimais. Veja na figura 30 o resultado da consulta.

		Trunc
1		45

**Figura 30:** TRUNC(45.84).

O caso mais complexo é corte com passagem de parâmetros negativos. Neste caso você está solicitando que seja cortado a dezena, centena, milhar. Para entender melhor teste a QUERY:

```
SELECT TRUNC(45.84,-1) "Round" FROM DUAL;
```

Para entender melhor, imagine que a *QUERY* desloca a casa decimal temporariamente. Assim você tem *TRUNC(4.584, 0)*. Neste caso 4 será mantido, e o número 584 será cortado. Como teve o deslocamento de uma casa o número será incrementado com zero e a operação de deslocado deve ocorrer à esquerda, deve-se voltar a casa decimal para sua posição. Assim o resultado é 40 como demonstra a figura 31.

Trunc	
1	
	40

**Figura 31:** *TRUNC(45.84, -1)*.

## POWER

A função **POWER** retorna à potência de uma base elevada a um expoente. A função apresenta a seguinte sintaxe:

```
POWER (n2, n1)
```

### POWER

função que retorna a potência de uma base elevada a um expoente.

Onde:

n2 é a base n2.

n1 é o expoente se n2 for negativo, então n1 deve ser um número inteiro.

Esta função toma como argumentos qualquer tipo de dado numérico ou qualquer tipo de dado não-numérico que pode ser convertido em um tipo de dado numérico. Se algum argumento for *BINARY\_FLOAT* ou *BINARY\_DOUBLE*, a função retornará *BINARY\_DOUBLE*. Caso contrário, a função retorna *NUMBER*. Teste o comando SQL para entender o funcionamento da função.

```
SELECT POWER(3,2) "Potência" FROM DUAL;
```

A consulta retorna 9 que é o resultado da potência de 32. Veja o resultado na figura 32.

Potência	
1	
	9

**Figura 32:** Função *POWER*.

Para ilustrar, imagine que a empresa queira criar um bônus com base no salário do empregado. Foi determinado que o bônus será dado pela formula: (salário/100)2. A QUERY ficaria assim:

```
SELECT first_name || ' ' || last_name NOME,  
       TO_CHAR(salary, 'FML09G999D00') SALARIO,  
       TO_CHAR(POWER((salary/100),2), 'FML09G999D00')  
             "Bônus"  
  FROM EMPLOYEES  
 WHERE department_id = (  
       SELECT department_id  
         FROM DEPARTMENTS  
        WHERE department_name = 'IT'  
 ) ;
```

O resultado da consulta é exibido na figura 33.

	NOME	SALARIO	Bônus
1	Alexander Hunold	R\$09.000,00	R\$08.100,00
2	Bruce Ernst	R\$06.000,00	R\$03.600,00
3	David Austin	R\$04.800,00	R\$02.304,00
4	Valli Pataballa	R\$04.800,00	R\$02.304,00
5	Diana Lorentz	R\$04.200,00	R\$01.764,00

**Figura 33:** Consulta com uso da função POWER.

Observe que a consulta retornou apenas os funcionários que eram do departamento TI com a proposta de bônus.

## SQRT

### SQRT

função que retorna a raiz quadrada do campo ou valor informado.

A função **SQRT** retorna a raiz quadrada do campo ou valor informado. A sintaxe é:

```
SQRT (n1)
```

Onde:

n1 é o número que se quer extrair a raiz quadrada.

Esta função apresenta como argumento qualquer tipo de dado numérico ou qualquer tipo de dado não-numérico que possa ser conver-

tido em um tipo de dado numérico. A função retorna o mesmo tipo de dado como o tipo de dado numérico do argumento. Se n1 for do tipo de dado *NUMBER*, o valor n1 não pode ser negativo, assim a *SQRT* retornará um número real. Caso n1 for um número binário ou ponto flutuante (*BINARY\_FLOAT* ou *BINARY\_DOUBLE*), então pode ser que  $n \geq 0$  o resultado será positivo, ou  $n = 0$ , o resultado será 0 ou  $n < 0$ , o resultado será NaN. Veja o código SQL

```
SELECT SQRT(81) "Raiz" FROM DUAL;
```

O resultado da QUERY é apresentado na figura 34.

RAIZ	
1	
	9

**Figura 34:** Função Raiz Quadrada.

Outro exemplo pode ser visto no SQL:

```
SELECT first_name || ' ' || last_name NOME,
       TO_CHAR(salary, 'FML09G999D00') SALARIO,
       TO_CHAR(SQRT(salary)*8, 'FML099D00') "Bônus Produtividade"
  FROM EMPLOYEES
 WHERE job_id =
       (SELECT job_id
        FROM JOBS
       WHERE job_title = 'Programmer'
      );
```

A consulta irá retornar todos os empregados que trabalham como programadores e serão listados o salário e o bônus de produtividade que é calculado pela raiz quadrada do salário do funcionário vezes oito. Veja o resultado na figura 35.

NOME	SALARIO	Bônus Produtividade
1 Alexander Hunold	R\$09.000,00	R\$758,95
2 Bruce Ernst	R\$06.000,00	R\$619,68
3 David Austin	R\$04.800,00	R\$554,26
4 Valli Pataballa	R\$04.800,00	R\$554,26
5 Diana Lorentz	R\$04.200,00	R\$518,46

**Figura 35:** Consulta com SQRT.

## MOD

### MOD

função que retorna o resto da divisão.

`MOD (n2, n1)`

Onde:

n2 é o número dividendo que sofrerá a divisão.

n1 é o número divisor.

*MOD* retorna o resto de n2 dividido por n1. Caso n2 for 0, a função irá retornar 0. Esta função tem como argumentos qualquer tipo de dado numérico ou qualquer tipo de dado não-numérico que pode ser convertido em um tipo de dado numérico. O SGBD determina o argumento com a maior precedência numérica, converte implicitamente os argumentos remanescentes para esse tipo de dado e retorna esse tipo de dado. Veja o exemplo a seguir:

```
SELECT MOD(3,2) "Resto" FROM DUAL;
```

O resultado da *QUERY* é apresentado na figura 36

RESTO	
1	1

**Figura 36:** MOD(3,2).

Note que dividir 3 por 2 sobra um resto, ou seja, sobra como resto da operação 1. O comando *MOD* é muito utilizado para retornar se determinado número é par ou ímpar. Todo número par retorna resto 0, enquanto um número ímpar retorna resto 1. Veja um segundo exemplo:

```

SELECT first_name || ' ' || last_name AS NOME,
       TO_CHAR(salary,'FML09G999D00') AS "Salário Base",
       TO_CHAR(commission_pct*33,'L09D00') AS "Comissão"
       TO_CHAR(MOD(salary, commission_pct*33),'L09D00')
AS "Bonificação",
       TO_CHAR(
salary+commission_pct*33+MOD(salary, commission_pct*33),
'FML09G999D00') AS "Salário"
FROM EMPLOYEES
WHERE commission_pct<>0;

```

A consulta retorna todos os funcionários que tem uma comissão e lista o salário base, o valor da comissão e uma bonificação que é calculada pelo resto da divisão do salário pela comissão. Por fim, a última coluna apresenta o salário final que é composto pelo salário base, comissão e bonificação. A figura 37 apresenta a consulta.

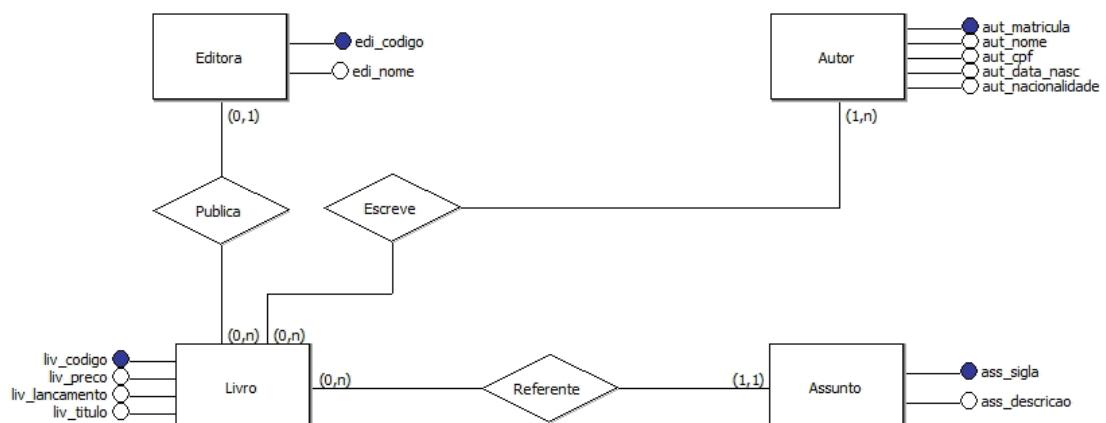
	NOME	Salário Base	Comissão	Bonificação	Salário
1	John Russell	R\$14.000,00	R\$13,20	R\$08,00	R\$14.021,20
2	Karen Partners	R\$13.500,00	R\$09,90	R\$06,30	R\$13.516,20
3	Alberto Errazuriz	R\$12.000,00	R\$09,90	R\$01,20	R\$12.011,10
4	Gerald Cambrault	R\$11.000,00	R\$09,90	R\$01,10	R\$11.011,00
5	Eleni Zlotkey	R\$10.500,00	R\$06,60	R\$06,00	R\$10.512,60
6	Peter Tucker	R\$10.000,00	R\$09,90	R\$01,00	R\$10.010,90
7	David Bernstein	R\$09.500,00	R\$08,25	R\$04,25	R\$09.512,50
8	Peter Hall	R\$09.000,00	R\$08,25	R\$07,50	R\$09.015,75
9	Christopher Olsen	R\$08.000,00	R\$06,60	R\$00,80	R\$08.007,40
10	Nanette Cambrault	R\$07.500,00	R\$06,60	R\$02,40	R\$07.509,00
11	Oliver Tuvault	R\$07.000,00	R\$04,95	R\$00,70	R\$07.005,65
12	Janette King	R\$10.000,00	R\$11,55	R\$09,25	R\$10.020,80
13	Patrick Sully	R\$09.500,00	R\$11,55	R\$05,90	R\$09.517,45
14	Allan McEwen	R\$09.000,00	R\$11,55	R\$02,55	R\$09.014,10
15	Lindsey Smith	R\$08.000,00	R\$09,90	R\$00,80	R\$08.010,70
16	Louise Doran	R\$07.500,00	R\$09,90	R\$05,70	R\$07.515,60
17	Sarath Sewall	R\$07.000,00	R\$08,25	R\$04,00	R\$07.012,25
18	Clara Vishney	R\$10.500,00	R\$08,25	R\$06,00	R\$10.514,25
	...				

**Figura 37:** Exemplo da Função MOD.

Nesta aula, você aprendeu funções que o SGBD oferece para tratar, importar e exibir dados. Aprendeu a *NVL* que trata os valores nulos, a *LOWER* para modificar uma *string* para minúscula, a função *UPPER* que converte uma *string* em maiúscula e *INITCAP* que torna a primeira letra de cada palavra maiúscula e as demais letras minúsculas. Viu como *RPAD* e *LPAD* trabalham juntas para gerar relatórios. Estudou a função *SUBSTR* que retorna parte de uma *string*, a função *INSTR* que retorna a posição de um caractere ou *substring* em um texto e a função *LENGTH* que nos dá a informação do tamanho de uma *string*. Comparou a função *TRANSLATE* e a função *REPLACE* que substituem caractere e *string* em uma expressão. Em se tratando de números, a função *ROUND* mostrou como arredondar um valor enquanto a função *TRUNC* corta parte dele. Você pode ver as funções *POWER* e *SQRT*, sendo que *POWER* eleva a potência e *SQRT* calcula a raiz quadrada. Por fim a função *MOD* lhe tornou possível calcular o resto de uma divisão.

## Atividades

- 01.** Com base no DER, resolva os exercícios:



- a.** Gere o diagrama lógico do Banco de Dados

---



---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

b. Crie o script com os comandos SQL.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

c. Insira os dados apresentados nas tabelas.

- d.** Exibir os nomes dos livros, que custam mais de R\$ 100,00 mostrando o as iniciais em maiúsculo e o valor.
- 
- 

- e.** Mostrar qual é o livro mais caro, mostrando somente o valor.
- 
- 

- f.** Contar quantas editoras existem.
- 
- 

- g.** Mostrar o nome do livro e data de lançamento no formato DD/MM/YYYY dos livros da editora com número 1.
- 
- 

- h.** Mostrar quantos autores existem no banco de dados.
- 
- 

- i.** Mostrar os nomes dos livros em maiúsculo.
- 
- 

- j.** Montar a seguinte expressão: "O livro XXX tem YYY caracteres", onde XXX será o nome do livro e YYY a quantidade de caracteres que este livro tem.
- 
- 
-

## Anotações



# AULA 9

## SQL AVANÇADO

### NESTA AULA

- » Junção de tabelas
- » Junção de tabelas com ALIAS
- » Operadores de junção de SQL

### ■ METAS DE COMPREENSÃO

- » Conhecer os conceitos de junção de tabelas
- » Aprender como realizar as junções com diferentes métodos
- » Compreender os operadores de junção do SQL

### ■ APRESENTAÇÃO

Nesta aula apresentaremos à você o conceito de junção de tabelas. Você vai conhecer alguns métodos e os operadores do SQL para junção de tabelas diferentes. Aprender a unir os dados que estão particionados em diferentes tabelas é uma das principais atividades realizada pelo DBA no SGBD. Você deve se dedicar a esta aula, visto que a maioria das entrevista e vagas de emprego exigem o conhecimento apresentado nesta aula. Por isso treine e faça todos os exercícios propostos ao final desta aula.

# ■ JUNÇÃO DE TABELAS

A capacidade de combinar as tabelas está entre as tarefas mais importantes de um banco de dados. As junções ocorrem quando você recupera dados de mais de uma tabela ao mesmo tempo. Para juntar tabelas basta na clausula FROM do comando SELECT nomear as tabelas que deseja. Quando você solicita duas ou mais tabelas o SGBD cria o produto cartesiano de todas as tabelas. O produto cartesiano será realizar a junção para cada linha da tabela A todas aos demais linhas da tabela B, até que todas as linhas de A recebam todas as linhas de B. Isso é útil em alguns cenários, mas a maioria das vezes você vai querer uma junção apenas dos valores correspondentes entre a tabela A e a tabela B. Para isso você verá diferentes técnicas que você pode adotar. Muitas das técnicas são semelhantes ou concorrentes em performance e se você é o DBA, cabe a você determinar qual será a utilizada na empresa. Contudo, quando você inicia sua carreira e entra em uma empresa, na maioria das vezes, algum DBA já definiu a política de união e junção de dados, cabendo a você conhecer a técnica utilizada para rapidamente se adaptar à empresa.

## JOIN

combina duas tabelas diferentes em uma consulta, a clausula INNER é opcional e se não for especificado o INNER estará implícito.

A maioria das junções ocorrem por meio de **JOINS** ou ligações. Uma ligação é usada quando a pesquisa SQL requer dados de mais de uma tabela do banco de dados. As linhas em uma tabela devem ser ligadas a outras linhas de acordo com o valor comum existente na coluna correspondente. Existem três tipos principais de condições de ligações: INNER-JOIN, NON-EQUI-JOIN e OUTER-JOIN. Através da clausula WHERE você consegue relacionar ou fazer as ligações da maioria das JOIN.

As JOIN tem como base a definição de que a modelagem conceitual e lógica foram impecáveis e assim todos os campos que são comuns são do mesmo tipo de dados e apresentam o mesmo nome nas tabelas que se pretende fazer a junção. Se os nomes dos campos chave primaria de uma tabela e chave estrangeira na outra tabela e os tipos de dados forem iguais então a consulta retorna com sucesso.

## NATURAL JOIN

A NATURAL JOIN é uma operação JOIN que cria uma cláusula de associação com base nas colunas comuns nas duas tabelas que estão

sendo juntas. Uma NATURAL JOIN pode ser uma associação INNER, LEFT OUTER ou RIGHT OUTER. O padrão é INNER JOIN. Caso você utilize o asterisco (\*) em uma consulta o retorno irá priorizar primeiro todas as colunas comuns nas tabelas, em seguida cada coluna na primeira tabela a esquerda na consulta que não é uma coluna comum e por fim cada coluna na segunda tabela a direita na consulta que não é uma coluna comum. Para exemplificar experimente:

```
SELECT department_name AS "Nome do Departamento",
       city AS "Cidade"
  FROM DEPARTMENTS
 NATURAL JOIN LOCATIONS;
```

O resultado da consulta é exibido na figura 01.

	Nome do Departamento	Cidade
1	IT	Southlake
2	Shipping	South San Francisco
3	Administration	Seattle
4	Purchasing	Seattle
5	Executive	Seattle
6	Finance	Seattle
7	Accounting	Seattle
8	Treasury	Seattle
9	Corporate Tax	Seattle
10	Control And Credit	Seattle
11	Shareholder Services	Seattle
12	Benefits	Seattle
...		

**Figura 01: NATURAL JOIN**

Observe que a coluna department\_name está na tabela DEPARTMENTS e a coluna city está presente na tabela LOCATIONS. Para que o SGBD consiga exibir os dados ele vai pesquisar quais colunas tem o mesmo nome e o mesmo tipo de dados em ambas as tabelas. A única coluna que apresenta o mesmo nome é a coluna location\_id. Na tabela LOCATIONS a coluna é chave primária e na tabela DEPARTMENTS a coluna é chave estrangeira. O comando NATURAL JOIN irá fazer uma WHERE automática onde a coluna location\_id deverá ser o item em comum que permite a união de ambas as tabelas. Com esses dados em

uma tabela temporária a consulta retorna as colunas solicitadas. Como pode ser visto o relacionamento entre as duas tabelas é do tipo INNER JOIN, mas não foi necessário explicitá-lo, pois, o NATURAL JOIN o fez de forma automática. O comando NATURAL JOIN lhe facilita, pois, todo o trabalho de relacionar as tabelas foi feito de forma automática.

## USING

Em alguns bancos de dados mais antigos às vezes o modelo não apresenta todas as colunas com nomes bem definidos. Pode ocorrer de algumas tabelas terem mais colunas com o mesmo nome e o mesmo tipo de dados sem serem as chaves primárias e estrangeiras. Esses bancos de dados podem conter colunas com o mesmo nome e o mesmo tipo de dados que você não queira como sendo pontos da relação entre as tabelas. Isso quer dizer que o comando NATURAL JOIN não poderá definir quais colunas devem ser utilizadas e nestes casos você vai querer determinar manualmente as colunas que irão participar. A cláusula **USING** deve ser utilizada, quando você tem várias colunas com o mesmo nome e precisa escolher o campo de junção. Ainda é um facilitador automático, mas permite que você informe o campo comum das tabelas que servirá para unificar os registros. Veja com o código SQL a seguir como utilizar a cláusula:

```
SELECT      department_name AS "Nome do Departamento",
            first_name || ' ' || last_name AS "Nome"
FROM        DEPARTMENTS
JOIN        EMPLOYEES
USING       (department_id)
WHERE       department_name = 'Sales';
```

**USING**  
pode ocorrer que duas tabelas tenham o mesmo nome e tipo de dados sem serem chaves primárias e estrangeiras, assim a cláusula define quais são os campos que serão utilizados para a junção. É um facilitador automático, mas permite que você informe o campo comum das tabelas.

O resultado da QUERY pode ser visto na figura 02.

	Nome do Departamento	Nome
1	Sales	John Russell
2	Sales	Karen Partners
3	Sales	Alberto Errazuriz
4	Sales	Gerald Cambrault
5	Sales	Eleni Zlotkey
6	Sales	Peter Tucker
7	Sales	David Bernstein
8	Sales	Peter Hall
9	Sales	Christopher Olsen
10	Sales	Nanette Cambrault
11	Sales	Oliver Tuvault
12	Sales	Janette King
13	Sales	Patrick Sully
14	Sales	Allan McEwen
...		

**Figura 02: USING**

Neste exemplo a tabela EMPLOYEES e a tabela DEPARTMENTS tem duas colunas em comum. A coluna department\_id é chave primaria na tabela DEPARTMENTS e chave estrangeira na tabela EMPLOYEES. Contudo, a coluna manager\_id armazena quem cheia os departamentos da empresa enquanto que na tabela EMPLOYEES ela armazena quem é o chefe do empregado. Assim, se você quer que a consulta traga todos o nome de todos os empregados que tem trabalham no departamento de vendas da empresa será necessário utilizar a clausula USING. Com ela a QUERY retorna 34 linhas, se você tivesse feito a consulta com NATURAL JOIN, a consulta retornaria 6 linhas, ou seja, apenas o nome dos funcionários que são chefe no departamento vendas.

## EQUI-JOIN

Uma **EQUI-JOIN** é uma junção que se realiza na WHERE com uma condição de junção contendo um operador de igualdade. A EQUI\_JOIN combina linhas de uma tabela associada a uma ou mais linhas em outra tabela com base na igualdade de valores. A sintaxe é:

```
SELECT colunas
FROM tabela_1, tabela_2
WHERE tabela_1.coluna = tabela_2.coluna;
```

### EQUI-JOIN

é uma junção realizada na clausula WHERE onde existe a condição de junção por meio do operador de igualdade. Combina linhas de uma tabela com uma ou mais linhas em outra tabela com base na igualdade de valores apresentados na clausula.

Onde:  
 colunas são as colunas que você deseja exibir  
 tabela\_1 e tabela\_2 são as tabelas que você irá relacionar  
 tabela\_x.coluna são as chaves primárias e chaves estrangeiras das  
 tabelas

O exemplo a seguir retorna o primeiro nome, o cargo de cada funcionário, o número e o nome do departamento em que o funcionário trabalha.

```
SELECT      first_name||' '||last_name AS "Nome do Funcionário",
            job_id AS "Cargo",
            DEPARTMENTS.department_id AS "Código do Departamento",
            department_name AS "Nome do Departamento"
        FROM EMPLOYEES, DEPARTMENTS
       WHERE EMPLOYEES.department_id = DEPARTMENTS.department_id
        ORDER BY 1;
```

Veja o resultado da QUERY na figura 03.

	Nome do Funcionário	Cargo	Código do Departamento	Nome do Departamento
1	Adam Fripp	ST_MAN		50 Shipping
2	Alana Walsh	SH_CLERK		50 Shipping
3	Alberto Errazuriz	SA_MAN		80 Sales
4	Alexander Hunold	IT_PROG		60 IT
5	Alexander Khoo	PU_CLERK		30 Purchasing
6	Alexis Bull	SH_CLERK		50 Shipping
7	Allan McEwen	SA_REP		80 Sales
8	Alyssa Hutton	SA_REP		80 Sales
9	Amit Banda	SA_REP		80 Sales
10	Anthony Cabrio	SH_CLERK		50 Shipping
11	Britney Everett	SH_CLERK		50 Shipping
12	Bruce Ernst	IT_PROG		60 IT
13	Charles Johnson	SA_REP		80 Sales
14	Christopher Olsen	SA_REP		80 Sales
...				

**Figura 03:** EQUI-JOIN

Note que a consulta apresenta três colunas que existem apenas na tabela EMPLOYEES, que são first\_name, last\_name e job\_id. Duas colunas têm como origem a tabela DEPARTMENTS, que são department\_id e department\_name. Na cláusula WHERE interliga-se as tabelas com a igualdade, por isso o nome EQUI-JOIN, onde existem colunas equivalentes ou iguais. Como a coluna department\_id existe em ambas as tabelas, você precisa informar a origem da tabela, por isso surgiu EMPLOYEES.department\_id e DEPARTMENTS.department\_id. Toda vez que você notar que as tabelas que estão sendo unidas nas consultas apresentam algumas colunas com o mesmo nome, será necessário informar o nome da tabela antes do nome da coluna, com a seguinte sintaxe: nome\_da\_tabela.nome\_da\_coluna. Por fim observe que a consulta tem a cláusula **ORDER BY** para garantir a ordem desejada, pois sem ele o comando SQL produz uma listagem em que as ordens das colunas não são relevantes, mas sim mais eficiente para ser gerado. Observe na figura 04 como a EQUI-JOIN processa a INNER JOIN.

### ORDER BY

instrução que solicita que os dados de retorno sejam ordenados tendo como referência as colunas apresentadas após o comando.

Nome do Funcionário	Cargo	Código do Departamento	Código do Departamento	Nome do Departamento
1 Adam Fripp	ST_MAN	50	1	10 Administration
2 Alana Walsh	SH_CLERK	50	2	20 Marketing
3 Alberto Errazuriz	SA_MAN	80	3	30 Purchasing
4 Alexander Hunold	IT_PROG	60	4	40 Human Resources
5 Alexander Khoo	PU_CLERK	30	5	50 Shipping
6 Alexis Bull	SH_CLERK	50	6	60 IT
7 Allan McEwen	SA_REP	80	7	70 Public Relations
8 Alyssa Hutton	SA_REP	80	8	80 Sales
9 Amit Banda	SA_REP	80	9	90 Executive
10 Anthony Cabrio	SH_CLERK	50	10	100 Finance
11 Britney Everett	SH_CLERK	50	11	110 Accounting
12 Bruce Ernst	IT_PROG	60	12	120 Treasury
...			...	



Nome do Funcionário	Cargo	Código do Departamento	Nome do Departamento
1 Adam Fripp	ST_MAN	50	50 Shipping
2 Alana Walsh	SH_CLERK	50	50 Shipping
3 Alberto Errazuriz	SA_MAN	80	80 Sales
4 Alexander Hunold	IT_PROG	60	60 IT
5 Alexander Khoo	PU_CLERK	30	30 Purchasing
6 Alexis Bull	SH_CLERK	50	50 Shipping
7 Allan McEwen	SA_REP	80	80 Sales
8 Alyssa Hutton	SA_REP	80	80 Sales
9 Amit Banda	SA_REP	80	80 Sales
10 Anthony Cabrio	SH_CLERK	50	50 Shipping
11 Britney Everett	SH_CLERK	50	50 Shipping
12 Bruce Ernst	IT_PROG	60	60 IT
13 Charles Johnson	SA_REP	80	80 Sales
14 Christopher Olsen	SA_REP	80	80 Sales
...			

**Figura 04:** Processo de INNER JOIN

Caso não especifique a clausula WHERE um produto cartesiano irá ocorrer entre as tabelas EMPLOYEES e DEPARTMENTS. Como a tabela DEPARTMENTS contém 27 linhas e a tabela EMPLOYEES contém 107 linhas, o produto cartesiano produzido será  $27 \times 107 = 2.889$  linhas. Cada linha da tabela EMPLOYEES será juntada com todas as linhas da tabela DEPARTMENTS. Você pode juntar quantas tabelas quiser em uma consulta, mas lembre-se de especificar uma condição de junção para cada par de tabelas. O número das condições na clausula WHERE será sempre o número de tabelas existentes no FROM menos um. Por exemplo se existir quatro tabelas no FROM vai existir três cláusulas de junção no WHERE. É importante lembrar que a maioria das cláusulas de junção são escritas com chave primária e a chave estrangeira das tabelas. Para exemplificar experimente o código SQL:

```
SELECT      first_name || ' ' || last_name AS "Nome do Funcionário",
            department_name AS "Nome do Departamento"
            city AS "Cidade"
FROM        EMPLOYEES, DEPARTMENTS, LOCATIONS
WHERE       EMPLOYEES.department_id = DEPARTMENTS.department_id
AND         LOCATIONS.location_id = DEPARTMENTS.location_id
AND         department_name IN ('IT', 'Finance')
ORDER BY    1;
```

O resultado da consulta é apresentado na figura 05

	Nome do Funcionário	Nome do Departamento	Cidade
1	Alexander Hunold	IT	Southlake
2	Bruce Ernst	IT	Southlake
3	Daniel Faviet	Finance	Seattle
4	David Austin	IT	Southlake
5	Diana Lorentz	IT	Southlake
6	Ismael Sciarra	Finance	Seattle
7	John Chen	Finance	Seattle
8	Jose Manuel Urman	Finance	Seattle
9	Luis Popp	Finance	Seattle
10	Nancy Greenberg	Finance	Seattle
11	Valli Pataballa	IT	Southlake

**Figura 05:** EQUI-JOIN com três tabelas

## JOIN OU INNER JOIN

O INNER JOIN seleciona todas as linhas de ambas as tabelas participantes, desde que ocorra uma correspondência entre as colunas. Um INNER JOIN é o mesmo que JOIN, combinando linhas de duas ou mais tabelas. Após ANSI SQL de 1992, a cláusula INNER é opcional se uma união for especificada e a junção não for especificado, INNER está implícito. A sintaxe é:

```
SELECT colunas  
FROM tabela_1  
[INNER] JOIN tabela_2  
ON tabela_1.coluna = tabela_2.coluna | USING cláusula;
```

Onde:

colunas são as colunas que você deseja exibir

tabela\_1 e tabela\_2 são as tabelas que você irá relacionar

tabela\_x.coluna são as chaves primárias e chaves estrangeiras das tabelas

cláusula é o nome da coluna que deve existir em ambas as tabelas

Você pode utilizar **ON** ou **USING** para realizar uma INNER JOIN. Para exemplificar veja como fica o exemplo anterior escrito com INNER JOIN.

```
SELECT      first_name || ' ' || last_name AS "Nome do Funcionário",  
            department_name AS "Nome do Departamento"  
            city AS "Cidade"  
FROM        EMPLOYEES  
INNER JOIN  DEPARTMENTS  
ON         EMPLOYEES.department_id = DEPARTMENTS.department_id  
INNER JOIN  LOCATIONS  
ON         LOCATIONS.location_id = DEPARTMENTS.location_id  
WHERE       department_name IN ('IT', 'Finance')  
ORDER BY    1;
```

### ON

referencia as colunas das tabelas que estão sendo unidas, determinando a cláusula de exclusão do produto cartesiano provocado pelo INNER JOIN.

Outra forma que poderia ser construída a consulta é:

```

SELECT      first_name||' '||last_name AS "Nome do Funcioná-
rio",
            department_name AS "Nome do Departamento"
            city AS "Cidade"
FROM    EMPLOYEES
INNER JOIN  DEPARTMENTS
INNER JOIN  LOCATIONS
ON      LOCATIONS.location_id = DEPARTMENTS.location_id
ON      EMPLOYEES.department_id = DEPARTMENTS.department_id
WHERE  department_name IN ('IT', 'Finance')
ORDER BY    1;

```

Ou ainda:

```

SELECT      first_name||' '||last_name AS "Nome do Funcioná-
rio",
            department_name AS "Nome do Departamento"
            city AS "Cidade"
FROM    EMPLOYEES
JOIN    DEPARTMENTS
USING      (department_id)
JOIN    LOCATIONS
USING      (location_id)
WHERE  department_name IN ('IT', 'Finance')
ORDER BY    1;

```

Todas são variações da mesma instrução e embora apresente uma sintaxe um pouco diferente uma da outra, são para o SGBD a mesma consulta. A cláusula ON pode fazer referência às tabelas que não estão sendo unidas. Contudo na maioria das vezes ela faz referência as tabelas que estão sendo juntas. O INNER JOIN junta as tabelas de acordo com a correspondência determinada como critério no operador ON.

Muitos estudantes de banco de dados se perguntam qual a diferença entre EQUI-JOIN e INNER JOIN. No EQUI-JOIN é na cláusula WHERE que todos os registros que correspondem à condição WHERE estão incluídos no conjunto de resultados. No INNER JOIN é feito o produto cartesiano dos dados e os dados que não correspondem à condição ON são excluídos do conjunto de resultados. A maioria dos DBAs gos-

tam da forma que o INNER JOIN separa as ligações entre as tabelas usando a cláusula ON e permitindo que o WHERE fique apenas como filtro dos elementos individuais da consulta. INNER JOIN é o padrão ANSI enquanto que EQUI-JOIN é mais orientada para o modelo relacional. O INNER JOIN é geralmente considerado mais fácil de ler e muitas empresas utilizam como padrão de consulta. O problema é que ele é um produto cartesiano das tabelas e isso pode tornar o processo um pouco mais lento especialmente quando você juntar muitas tabelas.

## ■ JUNÇÃO DE TABELAS COM ALIAS

Você já viu e vem utilizando o **ALIAS** para colocar nomes temporários em colunas substituindo o nome da coluna na consulta. O ALIAS também pode ser utilizado para identificar temporariamente o nome de uma tabela e ser utilizado junto aos comandos de junção. Em uma coluna você pode utilizar espaços em um ALIAS se você colocar aspas duplas, contudo esta prática não é recomendada para os nomes temporários de uma tabela. Lembre-se que o ALIAS só é válido dentro do escopo da instrução SQL. O uso do ALIAS simplifica a consulta e facilita a leitura do código SQL. É opcional utilizar a clausula AS para indicar um ALIAS para colunas, contudo seu uso é vetado para o nome temporário das tabelas. Experimente o código a seguir:

```
SELECT      e.first_name AS "Primeiro Nome",
            e.last_name AS "Sobrenome",
            d.department_name AS "Nome do Departamento"
FROM        EMPLOYEES e, DEPARTMENTS d
WHERE       e.department_id = d.department_id
AND         e.department_id = 30;
```

### ALIAS

substituir nomes de colunas e tabelas em uma consulta, nas colunas com objetivo de facilitar o entendimento dos dados apresentados e como medida de segurança ao banco de dados, nas tabelas para facilitar a construção de consultas extensas e com múltiplas tabelas.

Observe que o ALIAS na coluna fez uso da clausula AS para indicar o ALIAS, contudo, ao atribuir o ALIAS para o nome da tabela não é possível utilizar a clausula AS. O ALIAS escolhido foi o primeiro caractere do nome da tabela. Muitos DBAs adotam utilizar apenas uma letra para

indicar o apelido da tabela, mas não existe impedimentos para a utilização de mais caracteres. Note que você pode utilizar o ALIAS em todas as junções.

```
SELECT      e.first_name "Primeiro Nome",
            e.last_name "Sobrenome",
            d.department_name "Nome do Departamento"
FROM    EMPLOYEES e
JOIN    DEPARTMENTS d
ON      e.department_id = d.department_id
WHERE   e.department_id = 30;
```

O resultado das consultas com EQUI-JOIN e INNER JOIN são apresentados na figura 06.

	Primeiro Nome	Sobrenome	Nome do Departamento
1	Den	Raphaely	Purchasing
2	Alexander	Khoo	Purchasing
3	Shelli	Baida	Purchasing
4	Sigal	Tobias	Purchasing
5	Guy	Himuro	Purchasing
6	Karen	Colmenares	Purchasing

**Figura 06:** QUERY com ALIAS

## SELF-JOIN

### SELF-JOIN

combina a tabela com ela mesma, utilizado para consultas de associações unárias. Pode ser vista como uma associação de duas cópias temporárias da mesma tabela que é criado através do uso de ALIAS diferentes.

**SELF-JOIN** é uma auto-união, ocorre quando você associa uma tabela com ela mesma. Na modelagem de dados você viu as associações unárias. Neste momento, você deseja recuperar e consultar os dados de uma associação unária. A tabela deve possuir uma chave estrangeira que faz referência à sua própria chave primária. Para realizar a consulta você deve juntar a uma tabela com ela mesma, isso significa que cada linha da tabela é combinada com outra linha da tabela. A SELF-JOIN pode ser vista como uma associação de duas cópias da mesma tabela. A tabela não é realmente copiada, mas o SQL executa o comando como se fosse. A sintaxe do comando para juntar uma tabela com si é quase a mesma que para juntar duas tabelas diferentes. Para distinguir os nomes das colunas uns dos outros, são utilizados ALIAS para o nome da tabela atual, uma vez que ambas as tabelas têm o mesmo nome. Os ALIAS de nomes de tabelas são definidos na cláusula

FROM da instrução SELECT. Veja a sintaxe:

```
SELECT a.nome_coluna [, b.nome_coluna] [, ...]
FROM tabela_1 a, tabela_1 b
WHERE a.coluna_primaria = b.coluna_estrangeira;
```

Ou

```
SELECT a.nome_coluna [, b.nome_coluna] [, ...]
FROM tabela_1 a
JOIN tabela_1 b
ON a.coluna_primaria = b.coluna_estrangeira;
```

Onde:

nome\_coluna são as colunas que você deseja exibir

tabela\_1 a e tabela\_1 b é a mesma tabela com apelidos diferentes que você irá relacionar

x.coluna é as chaves primárias e chaves estrangeiras das tabelas

Para facilitar o entendimento você pode pesquisar a tabela EMPLOYEES onde employees\_id é a chave primária do empregado e manager\_id é a chave estrangeira de outro funcionário que é supervisor ou gerente. Se você quiser criar uma lista com os nomes dos funcionários e os nomes dos seus gerentes você terá de criar uma SELF-JOIN. Você pode criar a consulta com EQUI-JOIN como exemplificado a seguir:

```
SELECT      e.first_name||' '||e.last_name AS "Nome do Funcionário",
            m.first_name||' '||m.last_name AS "Nome do Gerente"
FROM        EMPLOYEES e, EMPLOYEES m
WHERE       e.manager_id = m.employee_id;
```

Outra forma é construindo uma INNER JOIN. O código a seguir exemplifica:

```
SELECT      e.first_name||' '||e.last_name AS "Nome do Funcionário",
            m.first_name||' '||m.last_name AS "Nome do Gerente"
FROM        EMPLOYEES e
JOIN      EMPLOYEES m
ON          e.manager_id = m.employee_id;
```

O resultado de ambas consultas é apresentado na figura 07.

	Nome do Funcionário	Nome do Gerente
1	Eleni Zlotkey	Steven King
2	Matthew Weiss	Steven King
3	Shanta Vollman	Steven King
4	John Russell	Steven King
5	Den Raphaely	Steven King
6	Karen Partners	Steven King
7	Kevin Mourgos	Steven King
8	Neena Kochhar	Steven King
9	Payam Kaufling	Steven King
10	Michael Hartstein	Steven King
11	Adam Fripp	Steven King
12	Alberto Errazuriz	Steven King
13	Lex De Haan	Steven King
14	Gerald Cambrault	Steven King
15	Jennifer Whalen	Neena Kochhar

...

**Figura 07: SELF-JOIN**

É muito comum fazer uma SELF-JOIN para comparar duas linhas na mesma tabela ou comparar dois intervalos de linhas dentro da mesma tabela. Para comparações grandes e complexas alguns defendem utilizar tabelas temporárias. Contudo, aqui estão as duas formas mais comuns de se juntar a uma tabela consigo mesma.

## NO-EQUIJOIN

Tem como base o relacionamento entre duas tabelas através de um grupo ou limites máximo e mínimos. Na maioria das vezes, não se trata de um relacionamento entre chave primária e chave estrangeira, ou relacionamento EQUI-JOIN. Todo o relacionamento que não é construído utilizando o operador de igualdade é denominado de **NO-EQUIJOIN**. Para exemplificar você deve criar a tabela JOB\_GRADES e inserir alguns registros. Para isso digite o SQL:

```
CREATE TABLE JOB_GRADES (
grade_level CHAR,
lowest_sal NUMBER(8),
highest_sal NUMBER(8),
CONSTRAINT grades_gra_pk PRIMARY KEY (grade_level)
);
```

```
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
```

**NO-EQUIJOIN**  
todo o relacionamento que não é construído utilizando o operador de igualdade.

```

VALUES ('A',1000,2999);
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
VALUES ('B',3000,5999);
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
VALUES ('C',6000,9999);
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
VALUES ('D',10000,14999);
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
VALUES ('E',15000,24999);
INSERT INTO JOB_GRADES (grade_level,lowest_sal,highest_sal)
VALUES ('F',25000,40000);

```

Agora você está pronto para testar NO-EQUIJOIN. O objetivo é relacionar o salário do empregado com a faixa uma faixa salarial e retornar à classificação do empregado. O funcionário que estiver com grade\_level 'A' está na menor faixa salarial da empresa, enquanto os funcionários que estiverem na faixa 'F' são os que estão na maior faixa salarial. A figura 08 ilustra o que se deseja em termo de relacionamento.

## EMPLOYEES

	NOME	SALARY
1	Steven King	24000
2	Neena Kochhar	17000
3	Lex De Haan	17000
4	Alexander Hunold	9000
5	Bruce Ernst	6000
6	David Austin	4800
7	Valli Pataballa	4800
8	Diana Lorentz	4200
9	Nancy Greenberg	12008
10	Daniel Faviet	9000
11	John Chen	8200
12	Ismael Sciarra	7700
13	Jose Manuel Urman	7800
14	Luis Popp	6900
	...	

## JOB\_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

O salário dos empregados deve estar entre o menor é o maior salário da tabela JOB GRADES

**Figura 08:** Relacionamento desejado em um NO-EQUIJOIN

Para realizar a consulta você pode utilizar o código SQL:

```

SELECT      e.first_name||' '||e.last_name AS "Nome do Funcionário",
            TO_CHAR(e.salary,'fm109G999D00') AS "Salário",
            j.grade_level AS "Faixa Salarial"
FROM        EMPLOYEES e, JOB_GRADES j
WHERE       e.salary
BETWEEN     j.lowest_sal AND j.highest_sal
ORDER BY    1;

```

O código acima tem como base uma EQUIJOIN, contudo a NO-EQUIJOIN também pode ser construída tendo como base uma JOIN, como exemplificado a seguir:

```
SELECT      e.first_name||' '||e.last_name AS "Nome do Funcionário",
            TO_CHAR(e.salary,'fm109G999D00') AS "Salário",
            j.grade_level AS "Faixa Salarial"
FROM        EMPLOYEES e
JOIN        JOB_GRADES j
ON          e.salary
BETWEEN    j.lowest_sal AND j.highest_sal
ORDER BY   1;
```

A figura 09 apresenta o resultado da QUERY.

Nome do Funcionário	Salário	Faixa Salarial
1 Adam Fripp	R\$08.200,00	C
2 Alana Walsh	R\$03.100,00	B
3 Alberto Errazuriz	R\$12.000,00	D
4 Alexander Hunold	R\$09.000,00	C
5 Alexander Khoo	R\$03.100,00	B
6 Alexis Bull	R\$04.100,00	B
7 Allan McEwen	R\$09.000,00	C
8 Alyssa Hutton	R\$08.800,00	C
9 Amit Banda	R\$06.200,00	C
10 Anthony Cabrio	R\$03.000,00	B
11 Britney Everett	R\$03.900,00	B
12 Bruce Ernst	R\$06.000,00	C
13 Charles Johnson	R\$06.200,00	C
14 Christopher Olsen	R\$08.000,00	C
15 Clara Vishney	R\$10.500,00	D
16 Curtis Davies	R\$03.100,00	B
...		

**Figura 09:** Consulta NO-EQUIJOIN

### OUTER JOIN

retorna todas as linhas que satisfazem a equivalência igual a uma INNER JOIN, mas pode retornar linhas que não se relacionaram, que estão presentes em apenas uma das tabelas. Pode ser LEFT OUTER JOIN, RIGHT OUTER JOIN ou FULL OUTER JOIN.

### OUTER JOIN

Às vezes você vai querer exibir além dos registros cujos campos em comum estejam presentes nas duas tabelas, podendo exibir ainda os que faltam. Neste caso, você pode construir uma consulta mesmo que não exista a equivalência entre as tabelas, ampliando o resultado. Uma **OUTER JOIN** retorna todas as linhas que satisfazem a equivalência igual a uma INNER JOIN, mas pode retornar linhas que não se relacionaram. Por exemplo, a tabela DEPARTMENTS possui o departamento CONTRACTING que não possui nenhum empregado alocado. No caso de você criar

uma consulta INNER JOIN na tabela de EMPLOYEES e DEPARTMENTS o departamento 190 é o único que não aparecerá. A OUTER JOIN pode ser **LEFT OUTER JOIN**, **RIGHT OUTER JOIN** ou **FULL OUTER JOIN**.

## ■ OPERADORES DE JUNÇÃO DE SQL

### LEFT OUTER JOIN

A LEFT OUTER JOIN ou apenas LEFT JOIN é o argumento que permite unir duas tabelas solicitando que todos os dados de uma tabela participem do resultado independente de existir um relacionamento com a outra tabela. A figura 10 apresenta o esquema gráfico da teoria dos conjuntos. Observe que a tabela A e a tabela B participam da consulta. A intersecção é a INNER JOIN, contudo solicitamos que todos os elementos da tabela A também estejam presentes no resultado, mesmo que não exista um resultado equivalente na tabela B. Os dados das colunas de B que não se relacionam com A devem retornar em branco.

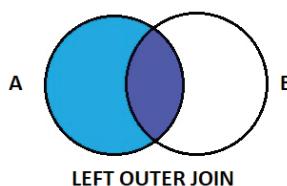


Figura 10: LEFT JOIN

Para escrever uma consulta que retorne os dados das tabelas A e B e retorna todas as linhas de A use a sintaxe LEFT [OUTER] JOIN na cláusula FROM, ou aplique o operador de junção externa (+) na cláusula WHERE. Veja o exemplo do comando:

```
SELECT      e.first_name || ' ' || e.last_name AS "Nome do Funcionário",
            e.department_id AS "Código do Departamento",
            d.department_name AS "Nome do Departamento"
FROM        EMPLOYEES e LEFT OUTER JOIN DEPARTMENTS d
ON          (e.department_id = d.department_id);
```

A QUERY faz a junção externa esquerda, todas as colunas de B na condição de associação e todas as de A. Para todas as linhas em A que

#### LEFT OUTER JOIN

argumento que permite unir duas tabelas solicitando que todos os dados de uma tabela participem do resultado independente de existir um relacionamento com a outra tabela. A consulta irá retornar todos os dados da tabela que está do lado esquerdo. Para saber qual é a tabela após comando FROM será LEFT seguido por um JOIN teremos a tabela RIGHT.

#### RIGHT OUTER JOIN

argumento que permite unir duas tabelas solicitando que todos os dados de uma tabela participem do resultado independente de existir um relacionamento com a outra tabela. A consulta irá retornar todos os dados da tabela que está do lado direito. Para saber qual é a tabela após comando FROM será LEFT seguido por um JOIN teremos a tabela RIGHT.

não possuem linhas correspondentes em B, o SGBD retorna nulo para todas as expressões de lista de seleção contendo colunas de B. Veja outra forma de fazer a LEFT JOIN em SQL:

```
SELECT          e.first_name || ' ' || e.last_name AS "Nome do Funcionário",
                e.department_id AS "Código do Departamento",
                d.department_name AS "Nome do Departamento"
FROM    EMPLOYEES e, DEPARTMENTS d
WHERE   e.department_id = d.department_id (+)
ORDER BY      d.department_name;
```

Veja o resultado que a consulta retorna na figura 11.

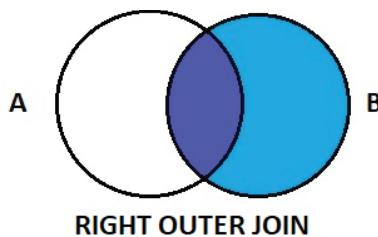
	Nome do Funcionário	Código do Departamento	Nome do Departamento
90	Peter Tucker		80 Sales
91	Eleni Zlotkey		80 Sales
92	Gerald Cambrault		80 Sales
93	Alberto Errazuriz		80 Sales
94	Karen Partners		80 Sales
95	John Russell		80 Sales
96	Lex De Haan		90 Executive
97	Neena Kochhar		90 Executive
98	Steven King		90 Executive
99	Luis Popp		100 Finance
100	Jose Manuel Urman		100 Finance
101	Ismael Sciarra		100 Finance
102	John Chen		100 Finance
103	Daniel Faviet		100 Finance
104	Nancy Greenberg		100 Finance
105	William Gietz		110 Accounting
106	Shelley Higgins		110 Accounting
107	Kimberely Grant		(null) (null)

**Figura 11:** Resultado da LEFT JOIN

Observe a QUERY como ficou a sintaxe no FROM. Você tem duas tabelas, uma representa a tabela A e outra representa a tabela B. No exemplo EMPLOYEES é a tabela A e DEPARTMENTS é a B. A consulta retorna todos os dados da tabela que está do lado esquerdo ou LEFT da palavra JOIN, neste caso todos os dados de EMPLOYEES foram exibidos e os dados em DEPARTMENTS que não apresentam a relação aparecem como nulo. Assim estamos exibindo todos os funcionários e seus departamentos. Note que existe um funcionário, Kimberely Grant que não tem um departamento atribuído.

## RIGHT OUTER JOIN

A RIGHT OUTER JOIN ou apenas RIGHT JOIN é o argumento que permite unir duas tabelas solicitando que todos os dados de uma tabela participem do resultado independente de existir um relacionamento com a outra tabela. A figura 12 apresenta o esquema gráfico da teoria dos conjuntos. Observe que a tabela A e a tabela B participam da consulta. A intersecção é a INNER JOIN, contudo solicitamos que todos os elementos da tabela B também estejam presentes no resultado, mesmo que não exista um resultado equivalente na tabela A. Os dados das colunas de A que não se relacionam com B devem retornar em branco.



**Figura 12:** RIGHT OUTER JOIN

Para escrever uma QUERY que execute a associação externa das tabelas A e B e retorna todas as linhas de B, use a sintaxe RIGHT OUTER JOIN na cláusula FROM.

```
SELECT      e.first_name || ' ' || e.last_name AS "Nome do Funcionário",
            e.department_id AS "Código do Departamento",
            d.department_name AS "Nome do Departamento"
FROM        EMPLOYEES e RIGHT OUTER JOIN DEPARTMENTS d
ON          (e.department_id = d.department_id);
```

A consulta faz a junção externa direita, para todas as colunas de A na condição de associação na cláusula WHERE. Todas as linhas em B que não possuem linhas correspondentes em A, o SGBD retorna nulo para todas as expressões da lista de seleção contendo colunas de A. Outra forma de realizar a consulta é com o operador de junção externa (+), como demonstra o SQL a seguir:

```
SELECT      e.first_name || ' ' || e.last_name AS "Nome do Funcionário",
```

```

        e.department_id AS "Código do Departamento",
        d.department_name AS "Nome do Departamento"
    FROM    EMPLOYEES e, DEPARTMENTS d
    WHERE   e.department_id (+) = d.department_id
    ORDER BY          d.department_name;

```

A QUERY apresenta duas tabelas na clausula FROM. A tabela EMPLOYEES é representada na figura 12 e a tabela DEPARTMENTS é representada como B. A consulta retorna todos os dados da tabela que está do lado direito ou RIGHT da palavra JOIN, neste caso todos os dados de DEPARTMENTES foram exibidos e os dados em EMPLOYEES que não apresentam a relação aparecem como nulo. Assim estamos exibindo todos os departamentos cadastrados e seus funcionários. Note que existe vários departamentos que não tem um funcionário atribuído.

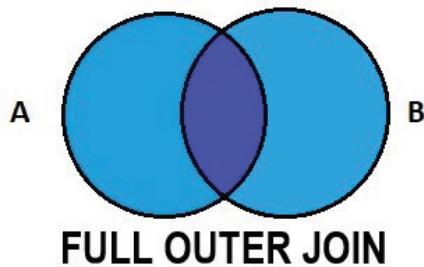
Veja o resultado que a consulta retorna na figura 13.

	Nome do Funcionário	Código do Departamento	Nome do Departamento
102	Daniel Faviet		100 Finance
103	Ismael Sciarra		100 Finance
104	Nancy Greenberg		100 Finance
105	William Gietz		110 Accounting
106	Shelley Higgins		110 Accounting
107			(null) Treasury
108			(null) Corporate Tax
109			(null) Control And Credit
110			(null) Shareholder Services
111			(null) Benefits
112			(null) Manufacturing
113			(null) Construction
114			(null) Contracting
115			(null) Operations
116			(null) IT Support
117			(null) NOC
118			(null) IT Helpdesk
119			(null) Government Sales
120			(null) Retail Sales
121			(null) Recruiting
122			(null) Payroll

**Figura 13:** Resultado RIGHT JOIN

## FULL OUTER JOIN

Para escrever uma consulta que executa uma associação externa e retorna todas as linhas de A e B, estendidas com nulos se não satisfizerem a condição de junção (uma associação externa completa), use a sintaxe FULL [OUTER] JOIN na cláusula FROM. A FULL JOIN retorna um RIGHT JOIN e um LEFT JOIN ao mesmo tempo. A figura 14 apresenta o esquema gráfico da teoria dos conjuntos.



**Figura 14:** RIGHT OUTER JOIN

Para escrever uma QUERY que execute a associação externa das tabelas A e B e retorna todas as linhas de A e todas as linhas de B, use a sintaxe FULL OUTER JOIN na cláusula FROM.

```
SELECT          e.first_name||' '||e.last_name AS "Nome do
Funcionário",
                e.department_id AS "Código do Departamento",
                d.department_name AS "Nome do Departamento"
FROM            EMPLOYEES a FULL OUTER JOIN DEPARTMENTS b
ON              (a.department_id = b.department_id);
```

Observe que a tabela A e a tabela B participam da consulta. A intersecção é a INNER JOIN, contudo solicitamos que todos os elementos das tabelas A e B estejam presentes no resultado, mesmo que não exista um resultado equivalente na tabela A ou na tabela B. Os dados das colunas de A que não se relacionam com B devem retornar em branco, e os dados das colunas de B que não se relacionam com A devem retornar em branco.

	Nome do Funcionário	Código do Departamento	Nome do Departamento
1	Steven King		90 Executive
2	Neena Kochhar		90 Executive
3	Lex De Haan		90 Executive
4	Alexander Hunold		60 IT
5	Bruce Ernst		60 IT
6	David Austin		60 IT
7	Valli Pataballa		60 IT
8	Diana Lorentz		60 IT
9	Nancy Greenberg		100 Finance
10	Daniel Faviet		100 Finance
...			
75	Ellen Abel		80 Sales
76	Alyssa Hutton		80 Sales
77	Jonathon Taylor		80 Sales
78	Jack Livingston		80 Sales
79	Kimberely Grant		(null) (null)
80	Charles Johnson		80 Sales
81	Winston Taylor		50 Shipping
82	Jean Fleaur		50 Shipping
83	Martha Sullivan		50 Shipping
84	Girard Geoni		50 Shipping
85	Nandita Sarchand		50 Shipping
...			
104	Susan Mavris		40 Human Resources
105	Hermann Baer		70 Public Relations
106	Shelley Higgins		110 Accounting
107	William Gietz		110 Accounting
108			(null) NOC
109			(null) Manufacturing
110			(null) Government Sales
111			(null) IT Support
112			(null) Benefits
113			(null) Shareholder Services
114			(null) Retail Sales
115			(null) Control And Credit
116			(null) Recruiting
117			(null) Operations
118			(null) Treasury
119			(null) Payroll
120			(null) Corporate Tax
121			(null) Construction
122			(null) Contracting
123			(null) IT Helpdesk

Figura 15: FULL OUTHER JOIN

Todas as consultas OUTER JOIN são úteis para retornar lacunas que apresentam dados esparsos. Essa associação é chamada de união externa particionada, pois é formada usando na query uma clausula que permite a exceção. Dados esparsos são dados que não possuem linhas para todos os valores possíveis de relacionar, como por exemplo departamento. Por exemplo, imagine a tabelas de vendas. Seria normal em determinado dia não existir nenhum dado registrado de venda para um produto. O preenchimento de lacunas de dados é útil em situações em que a escassez de dados complica a computação analítica ou onde alguns dados podem ser perdidos se os dados escassos forem consultados diretamente.

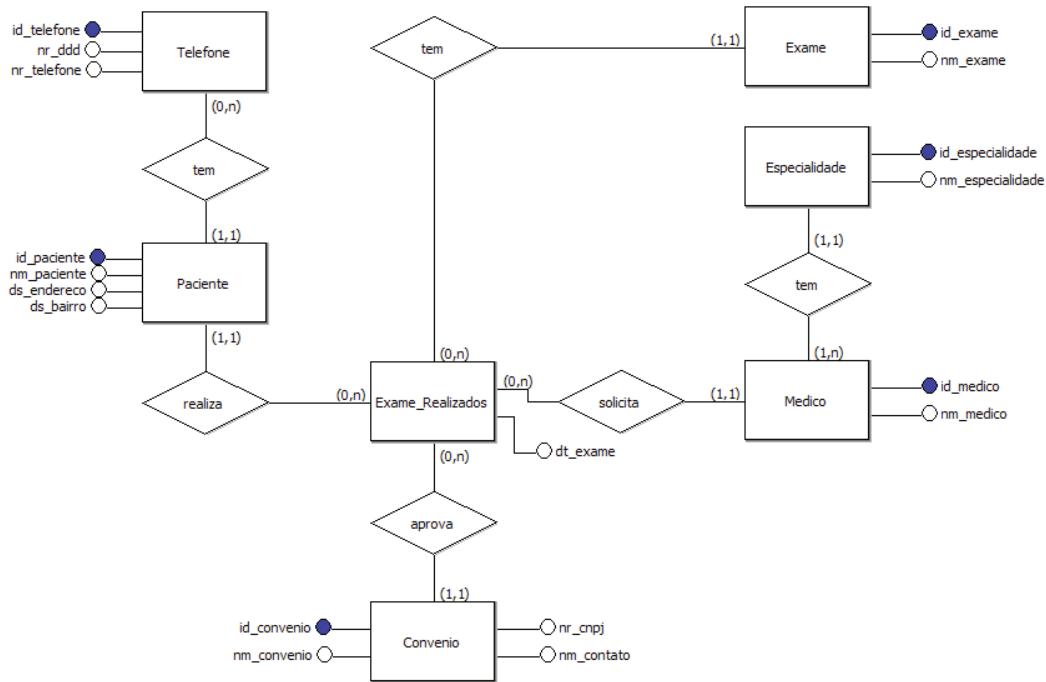


## síntese

Nesta quarta aula, você aprendeu os comandos para realizar junção de tabelas com diferentes métodos. Contudo, mesmo com o conceito em mente, ele só se tornará automático e transparente após você treinar muito. A escolha de qual método você irá utilizar irá depender da empresa, modelo e cenário que você se encontra. Por exemplo é comum ser contratado e encontrar um banco de dados operando, com um DBA e as escolhas dos métodos adotados na empresa já foram estabelecidos, por isso, é importante conhecer e estar preparado para utilizar todos os métodos de consulta. No começo você poderá sentir a necessidade de consultar o material diversas vezes, mas quanto mais utilizar o SQL, menos consultas serão necessárias. Não desanime ao realizar consultas complexas com diversas tabelas!

## Atividades

01. Tendo como referência o DER apresentado na figura 16, elabore os comandos SQL solicitados.



**Figura 16:** Diagrama da Clínica Particular Dra. X

**Observe as regras:**

Convênio

id\_convenio – numérico(6) – campo chave

nm\_convenio - varchar2(50) – obrigatório

nr\_cnpj – varchar2(25) – obrigatório

nm\_contato – varchar2(50)

Exame

id\_exame – numérico(6) – campo chave

nm\_exame – varchar2(50) – obrigatório

Exames\_Realizados

id\_exame\_realizado – num.(6) – chave

ds\_diagnostico – varchar2(255) – obrigatório

dt\_exame – data – obrigatório  
id\_medico – numérico(6) – obrigatório  
id\_convenio – numérico(6) – obrigatório  
id\_paciente – numérico(6) – obrigatório  
id\_exame – numérico(6) – obrigatório

Especialidade  
id\_especialidade – numérico(6) campo chave  
nm\_especialidade – varchar2(50) – obrigatório

Paciente  
id\_paciente – numérico(6) – campo chave  
nm\_paciente – varchar2 (50) – obrigatório  
ds\_endereco – varchar2(50)  
ds\_bairro – varchar2(40)

Telefone  
Id\_telefone – numérico(6) – campo chave  
nr\_ddd – numérico(2)  
nr\_telefone – varchar2(8)

Médico  
id\_medico – numérico(6) – campo chave  
nm\_medico – varchar2(50) – obrigatório  
id\_especialidade – numérico(6) obrigatório

- a. Gere o Modelo Lógico e Físico. Criar as tabelas com as restrições apresentada nas observações.
- 
- 
- 
- 
- 
- 
- 
-

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**b.** Insira 10 registros em cada tabela.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- c. Selecionar os médicos que tem especialidade OBSTETRA e que tenham sobrenome Silva no nome

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- d.** Exibir a quantidade de médicos existentes por especialidade, mostrando o nome da especialidade e a quantidade.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- e.** Exibir o nome do exame, data do exame dos convênios da AMIL

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- f.** Exibir todos os exames (Nome) que foram realizados pelo paciente TIBERCIO DOS SANTOS SAURO no mês de dezembro de 2017.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- g.** Exibir todos os médicos que tem a mesma especialidade do médico JOÃO DA SILVA SAURO

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- h.** Contar quantos exames foram realizados por Convênio, somente convênios que tem mais de 1 exames (Mostrar o nome do convênio e a quantidade).

---

---

---

---

---

---

---

---

---

- i.** Contar quantos pacientes fizeram Exames por convenio.

---

---

---

---

---

---

---

---

---

- j.** Exibir nome do paciente, nome do convênio, nome do médico, e nome do exame dos exames realizados em dezembro de 2017.

---

---

---

---

---

---

---

---

---

---

---

---

- k.** Exibir a quantidade de exames que são realizados por médico, ignorando o médico JOAO DA SILVA SAURO. Mostrar o nome do médico e a quantidade).

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- l.** Exibir a quantidade de exames que são realizados por mês, mostrando o nome do mês e o total de exames.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**m.** Exibir todos os pacientes que realizaram o exame de próstata.

---

---

---

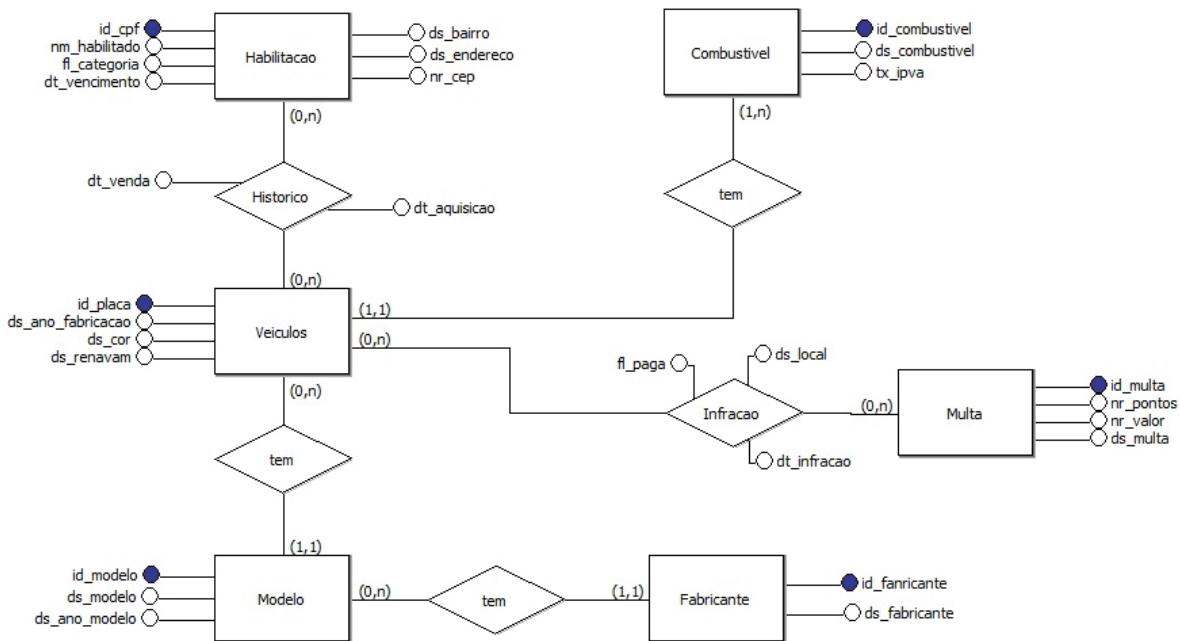
---

---

---

---

**02.** Tendo como referência o DER apresentado na figura a seguir, elabore os comandos SQL solicitados.



**Figura 17:** Diagrama Base do Departamento de Transito

Habilitacao

id\_cpf – varchar2(11) – campo chave

nm\_habilitado - varchar2(50) – obrigatório

ds\_endereco – varchar2(50) – obrigatório

nr\_cep – varchar2(8) – obrigatório

ds\_bairro – varchar2(40) – obrigatório  
fl\_categoria – char(1) – obrigatório  
dt\_vencimento – date – obrigatório

#### Historico

id\_cpf – varchar2(11) – campo chave  
id\_placa – varchar2(8) – campo chave  
dt\_aquisicao – date – obrigatório  
dt\_venda – date

#### Veículos

id\_placa – varchar2(8) – campo chave  
ds\_renavam – varchar2(13) – obrigatório  
ds\_ano\_fabric – numérico(4) – obrigatório  
ds\_cor – varchar2(40) – obrigatório  
id\_combustivel - numérico (2) – obrigatório  
id\_fabricante – numérico(4) – obrigatório

#### Modelo

id\_modelo – numérico(4) – campo chave  
ds\_modelo – varchar2(40) – obrigatório  
ds\_ano\_modelo – numérico(4) – obrigatório  
id\_fabricante – numérico(4) – obrigatório

#### Fabricante

id\_fabricante – numérico(4) – campo chave  
ds\_fabricante – varchar2(40) – obrigatório

#### Infração

id\_multa – numérico(6) – campo chave  
id\_placa – varchar2(8) – campo chave  
dt\_infracao – date – obrigatório  
ds\_local – varchar2(50) – obrigatório  
fl\_paga – varchar2(1) – aceitando somente S ou N

#### Multa

id\_multa – numérico(6) – campo chave

ds\_multa – varchar2 (50) – obrigatório  
nr\_pontos – numérico(2) – obrigatório  
nr\_valor – numérico(12,2) - obrigatório

Combustível

id\_combustivel - numérico(2) – campo chave  
ds\_combustivel – varchar2 (50) – obrigatório  
tx\_ipva – numérico(4,2) – obrigatório

- a. Criar as tabelas com os relacionamentos e regras de CONSTRAINT. Insira 2 proprietários de veículos e 7 infrações de trânsito distribuída entre os veículos.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- b.** Selecionar os veículos (Placa) que tem combustível GASOLINA e que foram fabricados em 2008.

---

---

---

---

---

---

---

---

- c.** Exibir a quantidade de condutores existentes por categoria, mostrando a categoria e o total de condutores.

---

---

---

---

---

---

---

---

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- d. Exibir o nome do proprietário, data de aquisição e data da venda do veículo com placa "BIB-1234"

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- e.** Exibir todas as infrações (Nome da Infração, Valor e Data da Infração) que foram cometidas pelo veículo "BIB-1234" no ano de 2016.

---

---

---

---

---

---

---

---

---

- f.** Exibir todos os veículos que tem o mesmo combustível do veículo DDD-7788

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- g.** Contar quantos veículos existem por tipo de combustível, somente dos combustíveis que tem mais de 100 ocorrências. (Mostrar o nome do combustível e a quantidade).

---

---

---

---

---

---

---

---

---

---

---

---

- h.** Exibir nome do proprietário, Placa do veículo, marca, modelo, ano de fabricação, combustível dos veículos adquiridos pelo CPF 11122233344

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- i. Exibir a quantidade de infrações por placa, ignorando o veículo XXX-9988. (Mostrar a placa, nome da infração e a quantidade).

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- j. Exibir a quantidade de infrações aplicadas por mês, mostrando o nome do mês e o total de infrações.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- k.** Criar uma visão, chamada visão01 que mostre o nome do proprietário, placa do veículo, data da infração, local da infração, descrição da multa e valor da multa cujo carro seja do condutor JOAO DA SILVA SAURO.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- l.** Exibir o total de multas a receber do ano 2016

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- m.** Calcular o valor total (taxa IPVA + multas) a pagar para o veículo  
BIB-1234

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- n.** Calcular os pontos na Habilitação do condutor JOAO DA SILVA  
SAURO

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Anotações

# AULA 10

## SQL PROCEDURAL

### NESTA AULA

- » Introdução ao PL/SQL
- » Blocos no PL/SQL
- » Declarações de variáveis
- » Declaração LOOP
- » Procedure
- » Cursor
- » Sequence
- » Function
- » Triggers

### ■ METAS DE COMPREENSÃO

- » Conhecer a linguagem de programação de banco de dados
- » Compreender PROCEDURE
- » Aprender a trabalhar com PL/SQL, Cursor, Sequence e TRIGGER

### ■ APRESENTAÇÃO

Nesta aula vamos apresentar à você a linguagem de programação existente no Banco de Dados Relacional. Você saberá como criar e executar PROCEDURES. O estudo de TRIGGER e PROCEDURE vai automatizar algumas tarefas. Com procedimentos e gatilhos você notará que algumas regras de negócio podem ser implementadas no SGBD.

# ■ INTRODUÇÃO AO PL/SQL

PL/SQL é uma linguagem orientada a objeto da Oracle para processamento e transações de alto desempenho. É uma linguagem integrada ao SQL que utiliza todas as instruções DDL, DML, funções e operadores do SQL. Ela suporta todos os tipos de dados SQL sem a necessidade de converter entre PL/SQL e os tipos de dados SQL. A PL/SQL suporta SQL estático e dinâmico além de permite que você execute uma consulta SQL e processe as linhas do conjunto de resultados uma de cada vez. Uma vantagem de uso de PL/SQL é que o bloco de instruções é executado somente no SGBD, por isso, você diminui drasticamente a comunicação entre cliente e o server. A desvantagem é que o uso excessivo deste recurso, poderá aumentar a carga do SGBD sobrecarregando o processador do servidor de banco de dados. Outra vantagem do PL/SQL é que as instruções serão compiladas, transformando as cláusulas WHERE e VALUES em variáveis de ligação, melhorando o desempenho. Os códigos compilados são armazenados em memória cache e compartilhados entre os usuários. Esses programas executáveis podem ser invocados repetidas vezes, reduzindo tráfego na rede e melhora o tempo de resposta do servidor.

## DECLARE

bloco que permite fazer declarações das variáveis, constantes e outros elementos de código

## BEGIN...END

bloco responsável pelo comandos e instruções que serão executados

## EXCEPTION

bloco que permite manipular as exceções, é uma seção especialmente para capturar e tratar os erros gerados

# ■ BLOCOS NO PL/SQL

PL/SQL é uma linguagem estruturada em bloco. Um bloco PL/SQL é definido pelas palavras-chave **DECLARE**, **BEGIN**, **EXCEPTION** e **END**. A figura 01 apresenta a estrutura em blocos. Observe que existem três seções de uso que você poderá acrescentar código, visto que o bloco END não permite acrescentar linhas de código, sendo o comando final. O bloco DECLARE permite fazer declarações das variáveis, constantes e outros elementos de código, que podem então ser usados nos blocos. O bloco BEGIN é responsável pelo comandos e instruções que serão executados. O bloco EXCEPTION permite você manipular as exceções, é uma seção especialmente para capturar e tratar os erros gerados.



**Figura 01:** Estrutura de Blocos do PL/SQL

Observe que duas seções são obrigatórias para criar um bloco executável. Você não precisa declarar nada no bloco DECLARE, e você não precisa interceptar as exceções no bloco EXCEPTION. Um bloco é uma declaração executável, e você pode aninhar blocos dentro de outros blocos. Um exemplo é o clássico "Olá Mundo!". Observe que a seção executável contém uma chamada de procedimento **DBMS\_OUTPUT.PUT\_LINE** que serve para exibir um texto na tela.

```
SET SERVEROUTPUT ON;
BEGIN
    DBMS_OUTPUT.put_line ('Olá Mundo!');
END;
```

#### DBMS\_OUTPUT. PUT\_LINE

ou /\* Comentários  
\*/ - o primeiro criar  
comentário em uma  
linha o segundo em  
múltiplas linhas

Observe que existe a necessidade de habilitar a saída na tela, por isso o comando SET SERVEROUTPUT foi modificado para ON. No próximo exemplo vamos declarar uma variável do tipo VARCHAR2 (string) com um comprimento máximo de 100 bytes. DBMS\_OUTPUT.PUT\_LINE aceita a variável, em vez da cadeia literal, para exibição.

```
DECLARE
    minha_mensagem
    VARCHAR2 (100) := 'Segundo exemplo';
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE (minha_mensagem);  
END;
```

O próximo bloco de exemplo adiciona uma seção de exceção que captura qualquer exceção (WHEN OTHERS) que pode ser levantada e exibe a mensagem de erro, que é retornada pela função SQLERRM.

```
DECLARE  
    minha_mensagem  
    VARCHAR2 (100) := 'Exemplo com tratamento de erro!';  
BEGIN  
    DBMS_OUTPUT.PUT_LINE (minha_mensagem);  
EXCEPTION  
    WHEN OTHERS  
    THEN  
        DBMS_OUTPUT.PUT_LINE (SQLERRM);  
END;
```

O próximo exemplo monstra a capacidade PL/SQL de aninhar blocos dentro de blocos.

```
DECLARE  
    minha_mensagem  
    VARCHAR2 (100) := 'Bem';  
BEGIN  
    DECLARE  
        minha_mensagem2  
        VARCHAR2 (100) := minha_mensagem || ' Vindo';  
    BEGIN  
        DBMS_OUTPUT.PUT_LINE (minha_mensagem2);  
    END;  
EXCEPTION  
    WHEN OTHERS  
    THEN  
        DBMS_OUTPUT.PUT_LINE  
            (dbms_utility.format_error_stack);  
END;
```

Para criar um comentário de linha única coloque -- e todo o restante da linha irá ser ignorado. O código a seguir apresenta linhas com comentários.

```
DECLARE
    numero_tabelas number;
BEGIN
    -- inicio do processo
    SELECT count(*) into numero_tabelas
    FROM user_objects
    WHERE objects_type = 'TABLE'
    -- verifica e imprime a quantidade de tabelas
    DBMS_OUTPUT.PUT_LINE ('Quantidade de Tabelas: '|| numero_
tabelas);
END;
```

Para criar comentários com múltiplas linha comece com /\*, termina com \*/. Os comentários com múltiplas linhas podem abranger várias linhas, veja o exemplo que apresenta dois comentários multilinha.

```
DECLARE
    /* Variáveis simples */
    pi           NUMBER := 3.1415926;
    diametro     NUMBER := 15;
    area         NUMBER;
BEGIN
    /* Esta linha calcula a área de um círculo usando pi,
       sua relação entre a circunferência e o diâmetro.
       Depois que a área é calculada, o resultado é exibido.
    */
    area := pi * diametro ** 2;
    DBMS_OUTPUT.PUT_LINE ('A área é: ' || TO_CHAR(area));
END;
```



dica

Você pode usar comentários de múltiplas linhas para delimitar uma seção de código. Isso vai lhe ajudar a testar e encontrar erros em seu código. Ao fazê-lo, tenha cuidado para não criar comentários de múltiplas linhas aninhadas. Um comentário de múltiplas linhas não pode conter outro comentário de múltiplas linhas. No entanto, um comentário de múltiplas linhas pode conter um comentário de uma única linha.

## ■ DECLARAÇÕES DE VARIÁVEIS

Ao encontrar uma declaração de uma variável o SGBD aloca espaço de armazenamento para o tipo de dados especificado e cria uma referência tendo como base o nome que você atribuiu. No PL/SQL você precisa declarar os objetos antes de poder fazer referência a eles. As declarações podem aparecer no bloco declarativo, subprograma ou pacote. Uma declaração de variável deve especificar o nome e o tipo de dados. Pode também especificar um valor inicial. O tipo de dados pode ser qualquer tipo de dados PL/SQL que incluem os tipos de dados SQL. Cada caractere atribuído ao nome de uma variável é significativo, sendo ele alfabético ou não. Vale lembrar que o PL/SQL para o nome de variáveis será sensível a maiúsculas e minúsculas, mas como linguagem isso não ocorre, ou seja, ele não diferencia os comandos em maiúsculas e minúsculas, deixando o desenvolvedor livre para utilizar como pretender ou gostar. Ao dar um nome a uma variável, você deve garantir que não está utilizando uma das palavras reservadas ou um identificador predefinido do PL/SQL. Crie nome significativos que depois de meses ao ler o código faça sentido de maneira fácil e rápida. Pode colocar símbolos como # para números, \$ para dinheiro e \_ para separar palavras compostas. É importante lembrar que o nome não pode exceder a 30 caracteres. Experimente o código a seguir:

```

SET SERVEROUTPUT ON;
DECLARE
    a# INTEGER := 10;
    b# INTEGER := 20;
    c# INTEGER;
    f$ REAL;
BEGIN
    c# := a# + b#;
    DBMS_OUTPUT.PUT_LINE('O valor de c#: ' || c#);
    f$ := 70.0/3.0;
    DBMS_OUTPUT.PUT_LINE('O valor de f$: ' || f$);
END;

```

Para declarar uma constante você deve acrescentar a palavra-chave **CONSTANT** e atribuir um valor inicial. Em uma declaração de variável, o valor inicial é opcional, a menos que você especifique a restrição NOT NULL. Enquanto que numa declaração constante, o valor inicial é necessário. Para especificar o valor inicial, use o operador de atribuição := ou a palavra-chave **DEFAULT**, seguida de uma expressão. A expressão pode incluir constantes previamente declaradas e variáveis previamente inicializadas.

#### CONSTANT

palavra-chave que indica um endereço de memória que mantém um valor imutável atribuído

```

DECLARE
    horas_trabalhadas      INTEGER := 40;
    quantidade_funcionarios INTEGER := 0;
    pi                      CONSTANT REAL := 3.14159;
    diametro                REAL := 1;
    area                    REAL := (pi * diametro ** 2);

BEGIN
    NULL;
END;

```

#### DEFAULT

palavra-chave que indica um valor padrão que será atribuído automaticamente se nenhum outro o fizer

Se você não especificar um valor inicial para uma variável o SBGD irá atribuir NULL para a variável. Antes de utilizar uma variável é necessário atribuir um valor.

```
DECLARE
```

```

    contar INTEGER; -- valor inicial e NULL
BEGIN
    contar := contar + 1; -- NULL + 1 continua NULL

    IF contar IS NULL THEN
        DBMS_OUTPUT.PUT_LINE ('contar é NULL.');
    END IF;
END;

```

### %TYPE

indica que o item deve ter o mesmo tipo de dado de uma variável ou coluna declarada anteriormente ou declarada em uma tabela

Você pode utilizar o atributo **%TYPE** para declarar um item do mesmo tipo de dado de uma variável declarada anteriormente ou para declarar o item com o mesmo tipo de dado de uma coluna de uma tabela sem ser necessário consultar o tipo de dado. Caso a tabela ou item mude, a variável mudará de acordo. O item de referência herda o tipo do dado, o tamanho de dados e as restrições. O item de referência não herda o valor inicial. Portanto, tome cuidado ao referenciar uma variável que tenha restrição NOT NULL, você terá de especificar um valor inicial. O atributo %TYPE é particularmente útil ao declarar variáveis para manter os valores proveniente do banco de dados.

### DECLARE

```

nome      VARCHAR(25) NOT NULL := 'Smith';
seu_nome nome%TYPE := 'João';

BEGIN
    DBMS_OUTPUT.PUT_LINE ('Nome = ' || nome);
    DBMS_OUTPUT.PUT_LINE ('Seu nome = ' || seu_nome);
END;

```

Experimente outro código e observe como criar variáveis a partir de tipos definidos em uma tabela.

### DECLARE

```

codigo departments.department_id%type := 10;
nome departments.department_name%type;
localizacao locations.city%type;

BEGIN
    SELECT departments.department_name, locations.city

```

```

INTO nome, localizacao
    FROM departments
    INNER JOIN locations
    ON departments.location_id = locations.location_id
    WHERE departments.department_id = codigo;
    DBMS_OUTPUT.PUT_LINE (
'Departamento ' || nome || ' está na cidade de ' || localização
);
END;

```

Para referenciar um identificador, veja por exemplo o código:

```

DECLARE
    a INTEGER; -- Declaração
BEGIN
    a := 1;      -- Referência de forma direta
END;

```

## ■ DECLARAÇÃO LOOP

As instruções Loop executam as mesmas instruções com uma série de valores diferentes. As instruções do loop são: LOOP, **FOR LOOP**, Cursor FOR LOOP e WHILE LOOP. A estrutura básica de um LOOP é:

```

[nome] LOOP
    comandos
END LOOP [nome];

```

A cada iteração do LOOP, as instruções ou comandos são executados e o controle retorna ao início do laço. A instrução LOOP termina quando uma declaração de controle do loop leva para fora do laço ou quando ocorre uma exceção. Para evitar um loop infinito, pelo menos uma declaração deve transferir o controle fora do laço. As instruções que podem transferir o controle fora do loop são: **CONTINUE**, EXIT, GOTO e RAISE.

### FOR LOOP

executa um número de repetições finitas, ou seja, em um intervalo especificado de vezes

### CONTINUE

sai da iteração atual de um LOOP incondicionalmente e transfere o controle para a próxima iteração

```

DECLARE
    x NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Dentro do loop: x = ' || TO_
CHAR(x));
        x := x + 1;
        IF x > 3 THEN
            EXIT;
        END IF;
    END LOOP;
    -- Depois do EXIT, o controle volta a executar as linhas
    -- normalmente
    DBMS_OUTPUT.PUT_LINE ('Depois do loop: x = ' || TO_
CHAR(x));
END;

```

### EXIT WHEN

sai da iteração atual de um loop quando a condição da cláusula WHEN é verdadeira e transfere o controle para o final do loop atual

Você pode utilizar uma forma mais compacta para escrever o código com **EXIT WHEN**. A declaração EXIT WHEN sai da iteração atual de um loop quando a condição da cláusula WHEN é verdadeira e transfere o controle para o final do loop atual. A cada iteração a instrução EXIT WHEN tem sua cláusula avaliada. Se a condição não for verdadeira, a instrução EXIT WHEN não faz nada. Para evitar um loop infinito, deve ser possível em algum momento tornar a condição verdadeira, como no exemplo a seguir.

```

DECLARE
    x NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Dentro do loop: x = ' || TO_
CHAR(x));
        x := x + 1;
        EXIT WHEN x > 3;
    END LOOP;
    -- Depois do EXIT, o controle volta a executar as linhas
    -- normalmente

```

```

DBMS_OUTPUT.PUT_LINE ('Depois do loop:  x = ' || TO_
CHAR (x));
END;

```

No exemplo, a declaração EXIT WHEN dentro da instrução LOOP transfere o controle para o final do loop atual quando x é maior que 3. Observe que construímos um código mais compacto, mas logicamente equivalente. Você pode declarar LOOP aninhado, ou seja, dentro de um LOOP colocar outro LOOP. Para isso será necessário definir rótulos como apresentado no código a seguir:

```

DECLARE
    t NUMBER(3) := 1;
    b NUMBER(2) := 0;
    m NUMBER(3) := 0;

BEGIN
    << externo >>
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Tabuada do '||t);
        << interno >>
        LOOP
            b := b + 1;
            m := t * b;
            DBMS_OUTPUT.PUT_LINE (t||' x '||b||' =
' ||m);
            EXIT interno WHEN b=10;
        END LOOP interno;
        b := 0;
        t := t + 1;
        EXIT externo WHEN t>10;
    END LOOP externo;
END;

```

A instrução CONTINUE sai da iteração atual de um LOOP incondicionalmente e transfere o controle para a próxima iteração. Para ilustrar observe a instrução CONTINUE dentro da instrução LOOP.

```

DECLARE
    x NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Valor de x = ' || TO_CHAR(x));
        x := x + 1;
        IF x < 3 THEN
            CONTINUE;
        END IF;
        DBMS_OUTPUT.PUT_LINE ('...');

        EXIT WHEN x = 6;
    END LOOP;
END;

```

Ao executar a instrução você observa que o controle de execução é interrompido ao encontrar a instrução CONTINUE e inicia o laço novamente. No exemplo ele irá fazer isso enquanto x for menor que 3 e após atingir esse valor passa a completar todas as instruções do laço. Você pode otimizar o código anterior com a instrução CONTINUE WHEN, conforme apresentado no código que segue.

```

DECLARE
    x NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Valor de x = ' || TO_CHAR(x));
        x := x + 1;
        CONTINUE WHEN x < 3;
        DBMS_OUTPUT.PUT_LINE ('...');

        EXIT WHEN x = 6;
    END LOOP;
END;

```

Para executar laços dentro de um número finito de repetições você pode utilizar a instrução FOR LOOP. Esta instrução executa um número de repetições de um intervalo especificado. A declaração tem essa estrutura:

```
[nome] FOR variavel IN [ REVERSE ] valor_inicial..valor_final LOOP  
    comandos  
END LOOP [nome];
```

A cada iteração da declaração FOR LOOP, suas instruções são executadas, e sua variável é incrementado ou decrementada e o controle retorna ao topo do loop. A instrução FOR LOOP termina quando sua variável atinge o valor especificado, quando uma declaração transfere o controle do loop para fora do laço ou ocorre uma exceção.

```
BEGIN  
    FOR x IN 0..5 LOOP  
        DBMS_OUTPUT.PUT_LINE ('Valor de x = ' || TO_CHAR(x));  
    END LOOP;  
END;
```

Sem **REVERSE**, o valor da variável começa em `valor_inicial` e aumenta em um a cada iteração do loop até atingir o `valor_final`. Se o `valor_inicial` for maior do que o `valor_final`, as instruções nunca serão executadas. Com o uso da instrução **REVERSE**, o valor da variável começa no `valor_final` e diminui em um a cada iteração do laço até atingir `valor_inicial`.

#### REVERSE

o valor da variável  
começa em um valor  
final e diminui em  
um a cada iteração  
do LOOP até atingir o  
valor inicial

```
BEGIN  
    FOR x IN REVERSE 0..5 LOOP  
        DBMS_OUTPUT.PUT_LINE ('Valor de x = ' || TO_CHAR(x));  
    END LOOP;  
END;
```



#### atenção

Você pode utilizar as variáveis do FOR LOOP livremente dentro do laço de repetição, não há a necessidade de declarar com a instrução DECLARE. É importante observar que você não poderá utilizá-la fora do laço. Utilizar a variável não declarada fora da instrução FOR LOOP causa um erro. Se declarar uma variável com DECLARE seu escopo será global enquanto a existente no FOR LOOP será local, mantendo assim dois valores diferentes.

O exemplo a seguir o valor\_final da instrução FOR LOOP é uma variável cujo valor é determinado em tempo de execução.

```
CREATE TABLE mala_direta (
    emp_codigo NUMBER,
    email      VARCHAR2(50)
);

DECLARE
    contar NUMBER;
BEGIN
    SELECT COUNT(employee_id) INTO contar
    FROM employees;
    FOR i IN 1..contar LOOP
        INSERT INTO mala_direta (emp_codigo, email)
        VALUES(i, 'informe seu e-mail');
    END LOOP;
END;
```

A PL/SQL permite ainda que você estabeleça através de uma consulta SQL o controle do laço. Você pode usar um loop simples e processar as linhas do conjunto de resultados uma de cada vez. Poderá controlar o processo precisamente usando instruções individuais para executar a consulta, recuperar os resultados e finalizar o processamento.

```
DECLARE
    contar NUMBER := 0;
BEGIN
    FOR lista IN (
        SELECT * FROM employees
        WHERE employee_id < 110
        ORDER BY employee_id
    )
    LOOP
        contar := contar + 1;
        DBMS_OUTPUT.PUT_LINE('Posição: ' || contar ||
            ' Nome: ' || lista.first_name ||
```

```

        ' ' || lista.last_name);
END LOOP;
END;

```

A instrução **SELECT INTO** recupera valores de uma ou mais tabelas de banco de dados da mesma maneira que a instrução SELECT faz, e as armazena em variáveis. Para cada item que uma SELECT retorna deve haver uma variável compatível com o tipo de dado correspondente. O SQL não possui um tipo BOOLEAN, então a variável não pode ser deste tipo.

### SELECT INTO

recupera valores de uma ou mais tabelas de banco de dados e os armazena em variáveis

O próximo exemplo usa uma instrução SELECT INTO para atribuir um bônus variável de 10% ao salário do empregado cujo employee\_id é 100.

```

DECLARE
    bonus      NUMBER(8,2);
BEGIN
    SELECT salary * 0.10 INTO bonus
    FROM employees
    WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE('bonus = ' || TO_CHAR(bonus));
END;

```

Ao trabalhar com o PL/SQL você pode querer fornecer informações ao usuário. Para isso pode usar o comando **PROMPT** para exibir mensagens para o usuário. Sua sintaxe é simples e fácil de utilizar:

### PROMPT

comando para exibir mensagens para o usuário

```
PRO [MPT] [texto];
```

Onde está [texto] você deve colocar a mensagem que deseja exibir. Se você omitir o texto, será exibida uma linha em branco na tela do usuário.

### PROMPT

```

PROMPT Este script insere de forma segura os dados,
PROMPT na tabela Empregados.
PROMP
```

## ACCEPT

comando que recebe  
entrada do usuário e  
o armazena em uma  
variável e permite  
um certo nível de  
controle sobre o que  
o usuário digita

O método confiável para obter a entrada do usuário é solicitar valores usando os comandos **ACCEPT** e **PROMPT**. O comando **ACCEPT** recebe entrada do usuário e o armazena em uma variável e permite um certo nível de controle sobre o que o usuário digita. Podem surgir problemas quando você simplesmente coloca variáveis de substituição em seus scripts. Esses problemas podem ser evitados através do uso do comando **ACCEPT**, pois você especifica uma variável e consegue validar a resposta do usuário. O comando **ACCEPT** exibe o **PROMPT** para o usuário e espera que o usuário responda para atribuir a resposta à variável. O exemplo a seguir mostra o uso de **PROMPT** em conjunto com **ACCEPT**:

```
SET SERVEROUTPUT ON;
ACCEPT v_nome CHAR PROMPT 'Digite seu nome';

DECLARE
    v_linha VARCHAR2(40);
BEGIN
    v_line := 'Bem vindo '||'&v_nome';
    DBMS_OUTPUT.PUT_LINE (v_line);
END;
```

Veja outro exemplo:

```
SET SERVEROUTPUT ON;
ACCEPT empno NUMBER FORMAT '9999' PROMPT 'Digite o código do
empregado ';
DECLARE
    v_empno      NUMBER(6) := &empno;
    v_emp_salary NUMBER(8,2);
BEGIN
    SELECT salary INTO v_emp_salary FROM employees WHERE em-
ployee_id = v_empno;
    DBMS_OUTPUT.PUT_LINE ( 'O salário do funcionário '|| v_emp-
no ||
    ' é de '||v_emp_salary);
END;
```

Poderia apresentar um valor padrão, caso o usuário não informe o DEFAULT defini o valor automaticamente. A única mudança necessária é o comando DEFAULT e o valor.

```
ACCEPT empno NUMBER FORMAT '9999' DEFAULT '110' PROMPT 'Digite
o código do empregado ';

DECLARE
    v_empno      NUMBER(6) := &empno;
    v_emp_salary NUMBER(8,2);
BEGIN
    SELECT salary INTO v_emp_salary FROM employees WHERE em-
ployee_id = v_empno;
    DBMS_OUTPUT.PUT_LINE ( 'O salário do funcionário ' || v_emp-
no ||
    ' é de ' || v_emp_salary);
END;
```

Para entrar com uma data utilize o DATE com FORMAT, como apresentado no código abaixo:

```
ACCEPT nasceu DATE FORMAT 'yyyy' PROMPT 'Digite o ano que
você nasceu YYYY';

DECLARE
    v_nasceu NUMBER;
    v_idade NUMBER;
    v_ano_agora NUMBER := to_char(SYSDATE, 'yyyy');
BEGIN
    v_nasceu := '&nasceu';
    v_idade := v_ano_agora - v_nasceu;
    DBMS_OUTPUT.PUT_LINE ('Sua idade é ' || v_idade);
END;
```

Vale lembrar que você pode definir o FORMAT com 'dd' para dia 'mm' para mês e 'yy' para ano, utilizando uma máscara como 'dd/mm/yyyy'.

Para não exibir no PROMPT os caracteres que estão sendo digitados, coloque antes do PROMPT o comando HIDE. Para exemplificar vamos criar em uma variável CHAR chamada PWD e verificar a entrada.

```

ACCEPT pwd CHAR HIDE PROMPT 'Digite a senha';
DECLARE
    v_senha CHAR(10) := '123456';
BEGIN
    IF v_senha = '&pwd' THEN
        DBMS_OUTPUT.PUT_LINE ('Acertou Miserável!');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Errou!');
    END IF;
END;

```

### IF-THEN

instrução para executa ou ignora uma sequência de instruções, causa um desvio de fluxo

Note que o código além de apresentar a instrução ACCEPT também apresenta a instrução IF. A instrução **IF-THEN** executa ou ignora uma sequência de instruções, dependendo do valor de uma expressão booleana. Ela causa um desvio de fluxo e por isso é muito utilizada. A instrução IF-THEN no PL/SQL pode apresentar três sintaxes como apresentado a seguir.

```

IF condicao THEN
    comandos
END IF;

```

Uma instrução condicional de nível único que executa uma sequência de instruções com base em uma determinada condição. O próximo nível de extensão acima da declaração IF-THEN é apresentado a seguir:

```

IF condicao THEN
    comandos
ELSE
    comandos
END IF;

```

A estrutura condicional IF-THEN-ELSE fornece a capacidade de dar direção para FALSE ou NULL também. Para múltiplos resultados possíveis de uma condição, usamos a declaração IF-THEN-ELSIF-ELSE.

```
IF condicao THEN
```

```

comandos
ELSIF condicao THEN
    comandos
[ ELSIF condicao THEN
    comandos
]...
[ ELSE
    comandos
]
END IF;

```

Para exemplificar o IF-THEN experimente o código:

```

ACCEPT nasceu DATE FORMAT 'yyyy' PROMPT 'Digite o ano que
você nasceu YYYY';
DECLARE
    v_nasceu NUMBER;
    v_idade NUMBER;
    v_ano_agora NUMBER := to_char(SYSDATE, 'yyyy');
BEGIN
    v_nasceu := '&nasceu';
    v_idade := v_ano_agora - v_nasceu;
    IF v_idade<11 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é uma criança!');
    END IF;
END;

```

Para testar o IF-THEN-ELSE você pode alterar o código anterior acrescentando o desvio caso a condição não seja verdadeira.

```

ACCEPT nasceu DATE FORMAT 'yyyy' PROMPT 'Digite o ano que
você nasceu YYYY';
DECLARE
    v_nasceu NUMBER;
    v_idade NUMBER;
    v_ano_agora NUMBER := to_char(SYSDATE, 'yyyy');
BEGIN
    v_nasceu := '&nasceu';

```

```

v_idade := v_ano_agora - v_nasceu;
IF v_idade<11 THEN
    DBMS_OUTPUT.PUT_LINE ('Você é uma criança!');
ELSE
    DBMS_OUTPUT.PUT_LINE ('Você não é uma criança!');
END IF;
END;

```

É importante ressaltar que se um dos valores for NULL a construção IF-THEN-ELSE se comporta como se a condição avaliada for FALSE. Por último a estrutura IF-THEN-ELSIF-ELSE é apresentada.

```

ACCEPT nasceu DATE FORMAT 'yyyy' PROMPT 'Digite o ano que
você nasceu YYYY';
DECLARE
    v_nasceu NUMBER;
    v_idade NUMBER;
    v_ano_agora NUMBER := to_char(SYSDATE, 'yyyy');
BEGIN
    v_nasceu := '&nasceu';
    v_idade := v_ano_agora - v_nasceu;
    IF v_idade<11 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é uma criança!');
    ELSIF v_idade<20 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é jovem!');
    ELSIF v_idade<40 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é adulto!');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Você é adulto Pleno!');
    END IF;
END;

```

No exemplo abaixo, realizamos a mesma operação que o modelo anterior, utilizando várias instruções IF.

```

ACCEPT nasceu DATE FORMAT 'yyyy' PROMPT 'Digite o ano que
você nasceu YYYY';

```

```

DECLARE
    v_nasceu NUMBER;
    v_idade NUMBER;
    v_ano_agora NUMBER := to_char(SYSDATE, 'yyyy');

BEGIN
    v_nasceu := '&nasceu';
    v_idade := v_ano_agora - v_nasceu;
    IF v_idade<11 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é uma criança!');
    END IF;
    IF v_idade<20 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é jovem!');
    END IF;
    IF v_idade<40 THEN
        DBMS_OUTPUT.PUT_LINE ('Você é adulto!');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Você é adulto Pleno!');
    END IF;
END;

```

O conjunto de instruções tem uma série de problemas e não estamos nos referindo apenas ao fato de que uma idade menor que 11 anos satisfará várias condições. Você poderia corrigir esse problema por simplesmente reordenar as condições. No entanto, todas as três declarações ainda precisam ser avaliadas a cada execução. Ao usar a instrução ELSIF, como no primeiro exemplo, o programa ao encontrar uma condição verdadeira, irá interromper as avaliações restantes. Uma vez que a condição avaliada é verdadeira, a cláusula THEN é executada e o controle irá pular todas as demais instruções até o END IF. O ELSE final só é executado quando todas as condições são avaliadas como falsas. Cada cláusula ELSIF deve ter uma condição e uma cláusula THEN. A cláusula ELSE não possui condição e a palavra THEN é omitida. Observe que a cláusula IF ou ELSIF podem apresentar uma condição composta com o uso de AND. A condição composta pode conter tantas avaliações quanto necessário, desde que toda a avaliação final seja verdadeira ou falsa.

# ■ PROCEDURE

Assim como em outras linguagens de programação, você pode criar seus próprios procedimentos em PL/SQL. Um procedimento armazenado no banco de dados ou stored procedure é uma coleção de comandos SQL. Há vantagens de criar seus próprios procedimentos, por exemplo, por ser compilado o conjunto de comandos tem um ganho de performance, por estar armazenado no servidor diminui o tráfego na rede entre cliente e servidor, pode ser chamado a partir de um comando SQL qualquer permitindo que você personalize o banco e as instruções do mesmo, encapsula rotinas de uso frequente no próprio servidor e com isso aumenta a disponibilidade para todas as aplicações e por último por criar códigos reutilizáveis diminuir as chances de erros de código abaixando os custos de desenvolvimento. Em uma procedure você poderá executar uma ação específica que não retorna valores. Use a instrução CREATE PROCEDURE para criar um procedimento no Banco de Dados. Um procedimento é um grupo de instruções PL/SQL que você pode chamar por um nome. Um procedimento apresenta um método ou uma rotina que pode ser chamada e informa ao banco de dados quais conversões de tipo devem ser feitas para os argumentos e valor de retorno. Os procedimentos oferecem vantagens nas áreas de desenvolvimento, integridade, segurança, desempenho e alocação de memória. A sintaxe para criar um procedimento é:

```
CREATE [OR REPLACE] PROCEDURE nome_procedure
[ (parametro [,parametro]) ]
IS
[declaracao_variaveis]
BEGIN
instrucoes
[EXCEPTION
Tratamento_erro]
END [nome_procedure];
```

Quando você cria um procedimento ou função, você pode definir parâmetros. Existem três tipos de parâmetros que podem ser declarados:

**IN** - parâmetro pode ser referenciado no procedimento. O valor do

parâmetro não pode ser substituído no procedimento. Existira uma passagem de valor do ambiente chamador para o procedimento e esse valor não pode ser alterado dentro do subprograma.

**OUT** - parâmetro não pode ser referenciado no procedimento, mas o valor do parâmetro pode ser substituído no procedimento. O dado irá sair do procedimento passando um valor para o ambiente chamador.

**IN OUT** - parâmetro pode ser referenciado no procedimento e o valor do parâmetro pode ser substituído no procedimento. Existira a passagem de valor do ambiente chamador para o procedimento. Esse valor pode ser alterado dentro da PROCEDURE e retornado com o valor atualizado para o ambiente chamador.

Para criar um procedimento em um esquema, você deve ter o privilégio CREATE PROCEDURE. Para criar um procedimento no esquema de outro usuário, você deve ter o privilégio CREATE ANY PROCEDURE. Para substituir um procedimento em outro esquema, você deve ter o privilégio de sistema ALTER ANY PROCEDURE.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE nr_telefone (
fone IN VARCHAR2
) IS
    novo_telefone VARCHAR2(20);
BEGIN
    novo_telefone := '(' || SUBSTR(fone,1,2) || ') '
                  || SUBSTR(fone,3,5) || ' - '
                  || SUBSTR(fone,8,4);
    DBMS_OUTPUT.PUT_LINE(novo_telefone);
END nr_telefone;
```

Para executar a PROCEDURE digite:

```
EXEC nr_telefone(11994911234);
```

Ao executar o procedimento observe que tratamos o número do telefone para sair no formato adotado no país em anos recentes, isolando os dados dos padrões de exibição. A chamada do procedimento passa apenas um parâmetro de entrada ou IN. Se você passar uma variável

para um subprograma como um parâmetro OUT ou IN OUT, e o subprograma atribui um valor ao parâmetro, a variável retém esse valor após o subprograma terminar. O exemplo a seguir passa a variável salario para o procedimento ajuste\_salario.

```
DECLARE
    salario_empregado NUMBER(8,2);
    codigo_empregado NUMBER := 100;

    PROCEDURE ajuste_salario (
        emp      NUMBER,
        salario IN OUT NUMBER,
        ajuste   IN NUMBER
    ) IS
    BEGIN
        salario := salario + ajuste;
    END;

BEGIN
    SELECT salary INTO salario_empregado
    FROM employees
    WHERE employee_id = codigo_empregado;

    DBMS_OUTPUT.PUT_LINE
        ('Antes de chamar a procedure, ajuste_salario: ' || salario_empregado);

    ajuste_salario (codigo_empregado, salario_empregado, 1000);

    DBMS_OUTPUT.PUT_LINE
        ('Depois de chamar a procedure, ajuste_salario: ' || salario_empregado);
END;
```

O procedimento atribui um valor ao parâmetro formal correspondente salario. Como salario é um parâmetro IN OUT, a variável salario retém o valor atribuído após o procedimento terminar. Para exemplificar experimente o código PL/SQL:

```

CREATE OR REPLACE PROCEDURE testa_parametros
(x IN NUMBER,
 y OUT NUMBER,
 z IN OUT NUMBER)
IS
BEGIN
    y := x * 2;
    y := y + z;
    z := y + x + z;
END testa_parametros;

SET SERVEROUTPUT ON;
DECLARE
    var1 NUMBER;
    var2 NUMBER;
    var3 NUMBER;
BEGIN
    var1 := 30;
    var2 := 40;
    var3 := 50;
    DBMS_OUTPUT.PUT_LINE ('Antes:' || var1 || '-' || var2 || '-'
    ' || var3);
    testa_parametros(var1,var2,var3);
    DBMS_OUTPUT.PUT_LINE ('Depois:' || var1 || '-' || var2 || '-'
    ' || var3);
END;

```

Observe os valores da saída e defina mentalmente como a procedure alterou os parâmetros. Caso tenha dificuldade faça um teste de mesa para determinar como a procedure alterou os valores.

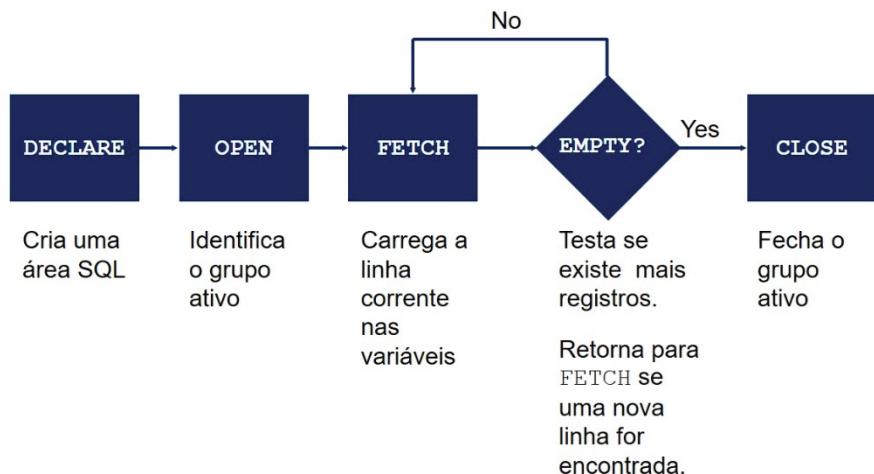


pense nisso

Quais seriam as vantagens de utilizar PROCEDURE em uma empresa. Se tratando de segurança quais rotinas você pode acrescentar em uma PROCEDURE instruções para evitar SQL INJECTION e o uso não autorizado por outros usuários. Também é possível criar PROCEDURE para evitar o acesso as tabelas e dados do sistema de forma direta.

# CURSOR

Cursor são áreas de memória compostas de linhas e colunas que servem para armazenar o resultado de uma SELECT. Normalmente a consulta de um cursor retorna várias linhas mas podem existir casos que retorna zero. Você usa um cursor para manipular ou processar linha por linha da consulta feita na SELECT. Existe dois tipos de cursores: implícitos e explícitos. Os implícitos são declarados e gerenciado pelo banco de dados enquanto os explícitos são declarados e gerenciado pelo programador. Ao declarar ou definir um cursor explícito você deverá dar-lhe um nome e associando-o a uma consulta. Para processar o conjunto de resultados da consulta você precisa abrir o cursor com a instrução OPEN, carregar o registro com a instrução FETCH e fechar o cursor com a instrução CLOSE. A figura 02 apresenta o graficamente as operações.



**Figura 02:** Operações necessárias para trabalhar com cursor

Você pode declarar um cursor e depois defini-lo mais tarde, mas isso dá mais trabalho e é pouco usual. A forma mais comum é declarar e defini-lo ao mesmo tempo. A declaração de cursor, que declara o cursor e o define tem a seguinte sintaxe:

```
CURSOR nome_do_cursor [parametros] [RETURN tipo_retorno]  
IS select;
```

Experimente o exemplo a seguir:

```
DECLARE
    CURSOR c1 IS
        SELECT first_name, last_name, salary
        FROM employees
        WHERE employee_id<110;
        v_nome employees.first_name%TYPE;
        v_sobrenome employees.last_name%TYPE;
        v_salario employees.salary%TYPE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO v_nome, v_sobrenome, v_salario;
        EXIT WHEN c1%NOTFOUND;
        IF v_valario<9000 THEN
            v_salario := v_salario * 0.01 + v_salario;
        END IF;
        DBMS_OUTPUT.PUT_LINE ('Nome: '||v_nome||' '||v_sobreno-
me ||
            'Novo Salário em R$'||_
            to_char(v_salario, '99,999.99');
    END LOOP;
    CLOSE c1;
END;
```

Depois de declarar e definir um cursor, para abri-lo utilizamos a instrução OPEN c1. Ao utilizar uma instrução OPEN o SGBD aloca recursos para processar a consulta, identifica o conjunto de resultados e posiciona o cursor antes da primeira linha dos resultados. Se o cursor tivesse variáveis ou parâmetros, seus valores afetariam o conjunto de resultados. Observe que fechamos o cursor com a instrução CLOSE c1, liberando assim os recursos do banco de dados. Depois de fechar um cursor, você não pode buscar registros ou fazer referência aos seus atributos. Tentar acessar um cursor que foi fechado acarreta um erro de exceção predefinido como INVALID\_CURSOR. Vale lembrar que você pode reabrir um cursor fechado, mas não pode abrir um cursor

que está aberto. Tentar abrir um cursor aberto gera a exceção predefinida CURSOR\_ALREADY\_OPEN. Depois de abrir o cursor, iniciamos a busca das linhas do conjunto de resultados com a instrução FETCH. A instrução FETCH retorna uma linha e sua sintaxe é:

```
FETCH nome_do_cursor INTO into_parametros;
```

O into\_parametros pode ser uma única variável ou uma lista de variáveis. Para cada coluna de retorno da consulta deve existir uma variável declarada em into\_parametros compatível com o tipo correspondente de dados. Os atributos %TYPE e %ROWTYPE são úteis para declarar variáveis e registros para uso em declarações FETCH. A instrução FETCH recupera uma linha de dados do conjunto de resultados, armazena os valores das colunas dessa linha nas variáveis e avança o cursor para a próxima linha. Normalmente, você usará a instrução FETCH dentro de uma instrução LOOP. Para detectar a condição de saída, usamos o atributo do cursor %NOTFOUND. Não existe exceção quando uma instrução FETCH não retorna nenhuma linha.

Você pode criar um cursor que possui parâmetros para passar diferentes parâmetros quando for abri-lo. Veja como fica o exemplo anterior com passagem de parâmetros.

```
ACCEPT v_dep NUMBER FORMAT '99' DEFAULT '20' PROMPT 'Digite o  
código do Departamento';  
  
DECLARE  
    CURSOR c1 (codigo_departamento NUMBER) IS  
        SELECT first_name, last_name, salary  
        FROM employees  
        WHERE employee_id=codigo_departamento;  
  
    PROCEDURE imprima_aumento IS  
        v_nome employees.first_name%TYPE;  
        v_sobrenome employees.last_name%TYPE;  
        v_salario employees.salary%TYPE;  
    BEGIN
```

```

LOOP
    FETCH c1 INTO v_nome, v_sobrenome, v_salario;
    EXIT WHEN c1%NOTFOUND;
    IF v_valario<9000 THEN
        v_salario := v_salario * 0.01 + v_salario;
    END IF;
    DBMS_OUTPUT.PUT_LINE ('Nome: '||v_nome||' '||v_sobrenome ||
                           'Novo Salário em R$'||to_char(v_salario, '99,999.99');
END LOOP;
END imprima_aumento;
BEGIN
    OPEN c1(&v_dep);
    DBMS_OUTPUT.PUT_LINE ('-----');
    DBMS_OUTPUT.PUT_LINE (' Aumento de Salário por código De-
partamento ');
    DBMS_OUTPUT.PUT_LINE ('-----');
    imprima_aumento;
    CLOSE c1;
END;

```

Uma maneira mais fácil de trabalhar com cursor é utilizar a instrução FOR LOOP. Ela permite que você execute uma instrução SELECT e percorra imediatamente as linhas do conjunto de resultados. Este comando usa um cursor implícito interno à sua estrutura. Depois de declarar as variáveis de índice de loop, a instrução FOR LOOP abre o cursor especificado automaticamente. A cada iteração, a instrução extrai uma linha do conjunto de resultados e a armazena. Quando não há mais linhas para buscar, a instrução fecha o cursor automaticamente. O cursor também fecha se o loop tiver seu controle interrompido. É possível utilizar um cursor sem declarar o comando CURSOR, como apresentado a seguir:

```

BEGIN
    FOR item IN (

```

```

        SELECT last_name, job_id
        FROM employees
        WHERE job_id LIKE '%PU_C%'
        ORDER BY last_name;
    )
LOOP
    DBMS_OUTPUT.PUT_LINE ('Nome: '||item.last_name||' Cargo:
'||item.job_id);
END LOOP;
END;

```

Também é possível utilizar a declaração explícita de um cursor para utilizar o cursor em outra parte do código. Se você for utilizar a instrução SELECT várias vezes defina um cursor explícito na instrução cursor FOR LOOP.

```

DECLARE
    CURSOR c1 IS
        SELECT last_name, job_id
        FROM employees
        WHERE job_id LIKE '%PU_C%'
        ORDER BY last_name;
BEGIN
    FOR item IN c1
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Nome: '||item.last_name||' Cargo:
'||item.job_id);
    END LOOP;
END;

```

Você pode declarar e definir em um cursor explícito com FOR LOOP passagem de parâmetros como apresenta o código PL/SQL a seguir.

```

ACCEPT cod_emp NUMBER FORMAT '999' DEFAULT '110' PROMPT 'Digite o código do Funcionário';
DECLARE
    CURSOR c1 (v_codigo_empregado employees.employee_id%TYPE) IS

```

```

SELECT last_name, job_id
  FROM employees
 WHERE employee_id = v_codigo_empregado;

BEGIN
  FOR item IN c1(&cod_emp)
    LOOP
      DBMS_OUTPUT.PUT_LINE ('Nome: '||item.last_name||' Cargo:
'||item.job_id);
    END LOOP;
END;

```

O cursor declarado com FOR LOOP pode fazer referência a colunas virtuais criadas por ALIAS.

```

BEGIN
  FOR item IN (
    SELECT first_name||' '||last_name AS nome, salary * 10 AS so-
    nho
      FROM employees
     WHERE ROWNUM <= 10
    ORDER BY sonho DESC
  ) LOOP
    DBMS_OUTPUT.PUT_LINE ('Nome: '||item.nome||
                           ' R$: ' ||to_char(item.so-
    nho, '999,999.99'));
  END LOOP;
END;

```

O interessante sobre cursor FOR LOOP é que ele abre, busca os dados e fecha o LOOP quando todas as linhas foram processadas.

## ■ SEQUENCE

Uma das frustrações dos administradores de banco de dados (DBAs) Oracle é não possuir incremento automático para uma coluna. Para

## SEQUENCE

campo de incremento automático usando o objeto de banco de dados

solucionar isso, você pode criar um campo de incremento automático usando o objeto de banco de dados **SEQUENCE**. O objeto gera um número inteiro único que pode ser atribuído nas chaves primárias. Quando um número de sequência é gerado, a sequência é incrementada, independentemente se a transação será ou não finalizada. Se dois usuários simultaneamente incrementarem a mesma sequência, os números gerados podem apresentar falhas ou saltos, porque os números de sequência estão sendo gerados por outro usuário. Um usuário nunca pode adquirir o número de sequência gerado para outro usuário. Depois de um valor de sequência ser gerado por um usuário, esse usuário pode continuar a acessar esse valor independentemente de a sequência ser incrementada por outro usuário. Os números de sequência são gerados independentemente de tabelas, portanto a mesma sequência pode ser usada para uma ou para várias tabelas. É possível que os números de sequência pareçam estar com problemas, porque eles podem pular, quando forem gerados e usados em uma transação que acabou não finalizada ou salva. Além disso, um único usuário pode não perceber que outros usuários estão acessando a mesma sequência. Use a instrução CREATE SEQUENCE para criar uma sequência, com a sintaxe mais usada é apresentada.

```
CREATE SEQUENCE nome_da_sequencia  
[START WITH inicial_valor]  
[INCREMENT BY incremento_valor] -- DEFAULT é 1  
[MINVALUE minimo_valor]  
[MAXVALUE maximo_valor];
```

A instrução START WITH define o primeiro número de sequência a ser gerado. Use esta cláusula apenas para iniciar uma sequência ascendente em um valor maior do que o mínimo ou para iniciar uma sequência descendente com um valor inferior ao máximo. Para as sequências ascendentes, o valor padrão é o valor mínimo. Para sequências descendentes, o valor padrão é o valor máximo. Esse valor inteiro pode ter 28 ou menos dígitos. MAXVALUE especifica o valor máximo que a sequência pode gerar. MAXVALUE deve ser igual ou superior a START WITH e deve ser maior que MINVALUE. Para uma sequência ascendente utiliza NOMAXVALUE para indicar valor máximo. A instrução

MINVALUE especifica o valor mínimo da sequência. MINVALUE deve ser inferior ou igual a START WITH e deve ser inferior a MAXVALUE. Especifique NOMINVALUE para indicar um valor mínimo numa sequência ascendente. Especifique CYCLE para indicar que a sequência continua a gerar novos valores depois de atingir o valor máximo ou mínimo. Depois que uma sequência crescente alcança seu valor máximo, ela gera seu valor mínimo. Depois que uma sequência descendente atinge o mínimo, gera seu valor máximo. Especifique NOCYCLE para indicar que a sequência não pode gerar mais valores após atingir seu valor máximo ou mínimo. Especifique CACHE para alocar e mantém na memória sequências para acesso mais rápido. Esse valor pode ter 28 ou menos dígitos. O valor mínimo para este parâmetro é 2. Você não precisa usar todos as instruções, veja como é fácil:

```
CREATE SEQUENCE dept_seq START WITH 280 INCREMENT BY 10;
```

Depois que uma sequência é criada, você pode acessar seus valores em uma instrução SQL com o nome\_sequencia.**NEXTVAL**. A instrução NEXTVAL irá incrementar a sequência e retornar o novo valor. Para exemplificar vamos inserir dois departamentos na tabela departments.

```
INSERT INTO departments VALUES (dept_seq.nextval, 'RH Brasil', null, 2800);
INSERT INTO departments VALUES (dept_seq.nextval, 'Vendas Brasil', null, 2800);
```

É possível consultar o valor corrente da sequência com nome\_sequencia.**CURRVAL**, como mostra o exemplo.

```
select dept_seq.currval from dual;
```

Você pode usar CURRVAL e NEXTVAL nos seguintes comandos: em uma SELECT que não está contida em SUBQUERY ou VIEW, em uma SUBQUERY contida em uma instrução INSERT, na cláusula VALUES de uma declaração INSERT e na cláusula SET de uma instrução UPDATE. Existe algumas restrições de uso de sequências. Você não pode usar CURRVAL e NEXTVAL nas seguintes construções: em uma SUBQUERY,

#### NEXTVAL

incrementa a sequência e retornar o novo valor

#### CURRVAL

consultar o valor corrente da sequência

em uma instrução DELETE, SELECT ou UPDATE, em uma VIEW, em uma instrução SELECT com o operador DISTINCT, em uma instrução SELECT com GROUP BY ou ORDER BY, em uma instrução SELECT que é combinada com outra instrução SELECT com o operador UNION, INTERSECT ou MINUS, na cláusula WHERE, no valor DEFAULT de uma coluna em uma instrução CREATE TABLE ou ALTER TABLE e em uma restrição CHECK.

## ■ FUNCTION

### FUNCTION

é um conjunto de instruções PL/SQL que você pode chamar pelo nome

Use a instrução CREATE **FUNCTION** para criar uma função. Uma função definida pelo usuário é um conjunto de instruções PL/SQL que você pode chamar pelo nome. As funções são muito semelhantes aos procedimentos, exceto que uma função retorna um valor ao ambiente em que é chamado. As funções do usuário podem ser usadas como parte de uma expressão SQL. A sintaxe é:

```
CREATE OR REPLACE FUNCTION nome_função
  (argumento1 modo tipo_de_dados,
   argumento2 modo tipo_de_dados,
   argumento3 modo tipo_de_dados)
  RETURN tipo_de_dado
  IS ou AS
    declarações
  BEGIN
    comandos
  EXCEPTION
  END nome_da_função;
```

Para exemplificar teste o código abaixo que cria uma função para converter um valor numérico em moeda real.

```
CREATE OR REPLACE FUNCTION to_real (valor IN NUMBER)
  RETURN VARCHAR2
  IS
```

```

    v_valor VARCHAR2(20);

BEGIN
    v_valor := 'R$ ' || to_char(valor,'99,999,999.99');
    RETURN v_valor;
END to_real;

```

Para testar a função utilize

```
SELECT to_real(1234.56) FROM dual;
```

Observe que a função definida pelo usuário é igual ao comportamento de um procedimento com uma diferença que a função retorna um dado. Veja o exemplo que tínhamos implementado como procedimento refeito como função.

```

SET SERVEROUTPUT ON;
CREATE OR REPLACE FUNCTION to_telefone (fone IN VARCHAR2)
RETURN VARCHAR2
IS
    novo_telefone VARCHAR2(20);
BEGIN
    novo_telefone := '(' || SUBSTR(fone,1,2) || ') '
                    || SUBSTR(fone,3,5) || ' - '
                    || SUBSTR(fone,8,4);
    RETURN novo_telefone;
END to_telefone;

```

Para testa a função

```
SELECT to_telefone(11994914656) FROM dual;
```

As funções definidas pelo usuário estão sujeitas às seguintes restrições: não podem ser usadas em uma CONSTRAINT CHECK ou uma cláusula DEFAULT em declaração CREATE TABLE ou ALTER TABLE. Além disso, quando uma função é chamada a partir de uma consulta DML, a função não pode ter os parâmetros OUT ou IN OUT. Não pode conter instruções COMMIT, ROLLBACK ou SAVEPOINT, portanto, po-

demos dizer que uma função definida pelo usuário não pode executar quaisquer instruções DDL.

## ■ TRIGGERS

Trigger ou gatilhos são blocos PL/SQL armazenados no banco de dados que ficam associado a uma tabela que disparados automaticamente sempre que ocorrer um evento ou ação associado a tabela como um INSERT, UPDATE ou DELETE. O SGBD executa automaticamente o gatilho quando as condições especificadas ocorrem. Use a instrução CREATE TRIGGER para criar e ativar um gatilho no banco de dados. Uma trigger pode estar associada a uma linha ou ROW LEVEL ou a uma tabela. Trigger pode ocorrer BEFORE - antes ou AFTER - depois que a ação tenha ocorrido. A grande maioria dos gatilhos que tenho observado utiliza a sintaxe mais básica, descrita abaixo.

```
CREATE [OR REPLACE] TRIGGER nome_da_trigger
{BEFORE | AFTER} evento_dml ON nome_da_tabela
[FOR EACH ROW]
[DECLARE]
BEGIN
    comandos
[EXCEPTION]
END;
```

Você deve escolher entre o método BEFORE ou AFTER, pois são obrigatórios. A cláusula FOR EACH ROW é opcional e definem o ponto de temporização para o gatilho. O evento\_dml pode ser um ou dos seguintes: INSERT, UPDATE ou DELETE. Para exemplificar teste a sequência de comandos abaixo.

```
CREATE OR REPLACE TRIGGER tg_departments
BEFORE
INSERT OR
UPDATE OF department_name, manager_id OR
```

```

DELETE
ON departments
BEGIN
CASE
WHEN INSERTING THEN
DBMS_OUTPUT.PUT_LINE ('Inserindo Departamento');
WHEN UPDATING ('department_name') THEN
DBMS_OUTPUT.PUT_LINE ('Alterando nome do Departamen-
to');
WHEN UPDATING ('manager_id') THEN
DBMS_OUTPUT.PUT_LINE ('Alterando gerente do Departamen-
to');
WHEN DELETING THEN
DBMS_OUTPUT.PUT_LINE ('Apagando Departamento');
END CASE;
END;

```

O evento desencadeante de um gatilho DML pode ser composto de várias instruções de disparo. Quando um deles dispara o gatilho, é possível determinar qual deles usando um desses predicados condicionais apresentado no exemplo. O próximo exemplo cria uma tabela de log e um gatilho que insere uma linha na tabela de log depois de qualquer declaração UPDATE afetar a coluna SALARY da tabela EMPLOYEES e após registrar o log atualiza SALARY.

```

DROP TABLE emp_log;
CREATE TABLE emp_log (
    emp_id      NUMBER,
    log_data    DATE,
    novo_salario NUMBER,
    acao        VARCHAR2(20)
);
CREATE OR REPLACE TRIGGER log_alteracao_salario
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN

```

```
INSERT INTO emp_log (emp_id, log_data, novo_salario, acao)
VALUES (:NEW.employee_id, SYSDATE, :NEW.salary, 'Novo Salario');
END;
```

Para testar a trigger utilize:

```
UPDATE employees
SET salary = salary + 1000.0
WHERE department_id = 20;
```

Quando você cria um gatilho, o banco de dados o habilita automaticamente. Você pode, posteriormente, desativar e ativar um gatilho com a cláusula DISABLE e ENABLE da instrução ALTER TRIGGER ou ALTER TABLE. Veja um exemplo de como desabilitar um gatilho.

```
ALTER TRIGGER tg_departments DISABLE;
```

Depois de ter criado um gatilho você pode achar que precisa removê-lo do banco de dados. Você pode fazer isso com a declaração DROP TRIGGER.

```
DROP TRIGGER tg_departments;
```

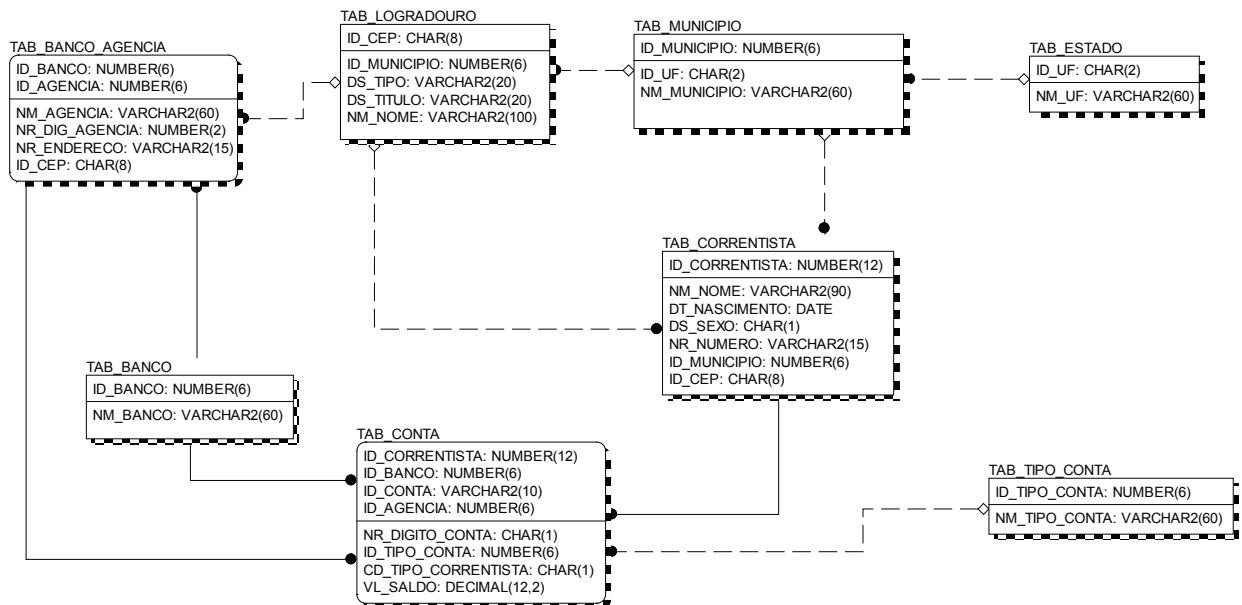
Este exemplo remove o gatilho do banco de dados.



### síntese

Nesta aula, você iniciou seus estudos na linguagem de programação PL/SQL. Aprendeu como ela trabalha com conceitos de blocos, viu como definir variáveis, solicitar e criar máscara via ACCEPT e PROMPT. Você pode ver e criar LOOP, explorar o controle a cada iteração com um cursor. Também foi apresentado o comando SEQUENCE que vai poder lhe ajudar no controle das chaves primárias. As possibilidades de criar blocos nomeados para reuso foram apresentadas com PROCEDURE e FUNCTION. Por fim, foi apresentada brevemente como atrelar suas rotinas às ações INSERT, UPDATE e DELETE nas tabelas com as TRIGGERS.

## Atividades



01. Crie um PROCEDURE PL/SQL para incluir dados na tabela tab\_correntista.
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
-

**02.** Faça um bloco PL/SQL para excluir um dado, onde o usuário digita o código do correntista a ser excluído.

---

---

---

---

---

---

---

---

**03.** Faça um bloco PL/SQL para que atualize o saldo de um determinado cliente. O saldo e o cliente deverão ser digitados pelo usuário.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# AULA 1

## Atividade 1

Crie um arquivo HTML que, utilizando JavaScript, armazene a base e a altura de um retângulo e calcule sua área e perímetro apresentando os resultados no corpo do HTML

### Expectativa de Resposta:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Atividade</title>
    <script>
        base = 2.45;
        altura = 5.32;
        area = base * altura;
        perimetro = (base * 2) + (altura * 2);
        document.write("A área é: " + area + "<br />");
        document.write("O perímetro é: " + perimetro + "<br />");
    </script>
</head>
<body>

</body>
</html>
```

## Atividade 2

Crie um HTML que permita criar uma lista de convidados para um churrasco. A lista de convidados deve sempre ficar visível no corpo da página. A página deve ter os botões de Inserir, Ordenar, Remover, Calcular Total de Carne. A pessoa deve inserir um nome em uma caixa de texto e clicar em Inserir o nome que será armazenado em um vetor. Ao clicar em ordenar, o vetor deve ser colocado em ordem alfabética. O botão Remover deve perguntar qual a posição a ser removido e eliminar o nome da pessoa da lista de convidados. O botão Calcular o total de carne deve contar o total de convidados e calcular para cada convidado 350g de carne bovina, 120g de linguiça e 30g de frango. Ao final do cálculo deve converter o resultado para quilos e exibir para o usuário no corpo da página.

# AULA 1

## Expectativa de Resposta:

```
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8" />
    <title>Churrasco 2</title>
    <script src="Lista.js"></script>
    <script>
        var convidados = new Lista();
        function cadastrar() {
            convidados.inserir(document.getElementById("nome").value);
            convidados.escrever();
        }
        function remover() {
            var nT = parseInt(prompt("Qual posição?"));
            convidados.retirarPosicao(nT);
            convidados.escrever();
        }
        function ordenar() {
            convidados.ordenar();
            convidados.escrever();
        }
        function calcular() {
            var quantidade = convidados.mlist.length;
            var carne = quantidade * 350 / 1000;
            var frango = quantidade * 30 / 1000;
            var linguica = quantidade * 120 / 1000;
            document.write("Carne: " + carne + "<br />");
            document.write("Frango: " + frango + "<br />");
            document.write("Liguica: " + linguica + "<br />")
        }
    </script>
</head>
<body>
    <form>
        <label>Nome: </label>
        <input type="text" id="nome" />
    </form>
    <input type="button" onclick="cadastrar();" value="Inserir" />
    <input type="button" onclick="remover();" value="Remover"/>
    <input type="button" onclick="ordenar();" value="Ordenar"/>
    <input type="button" onclick="calcular();" value="Calcular Total Carne"/>
    <div id="area"><h1>Lista de Convidados</h1></div>
</body>
</html>
```

# AULA 2

## Atividade 1

Quais as vantagens e desvantagens entre um sistema de processamento de arquivos e um sistema de gerenciamento de banco de dados?

### Expectativa de Resposta:

Vantagens do SGBD: proporcionar abstração isolando o usuário dos detalhes internos das estruturas de dados, aprimorar o compartilhamento, tornar fácil e rápido o acesso aos dados, proporcionar independência dos dados em relação a estratégia de acesso, facilitar para que diversas aplicações acessem a base de dados, bloquear o acesso a sua estrutura interna, permitir o acesso somente através de uma interface segura aplicando uma política de privacidade e segurança de dados, melhorar a integração de dados, minimizar a inconsistência de dados, aumentar a produtiva e aprimorar a tomada de decisão

Desvantagens do sistema de arquivos: necessidade de programação para buscas simples, necessário tempo para programar novas consultas, alterar as estruturas de dados em um sistema que está operacional é difícil, garantir compatibilidade quando ocorrem mudanças, dificuldade de administrar com crescimento do banco e dificuldade de implementar segurança.

## Atividade 2

Quais os tipos de arquiteturas de banco de dados, vantagens e desvantagens de cada uma?

### Expectativa de Resposta:

Arquitetura Mainframe – Vantagens: grande poder de processamento, tecnologia estável, suporte a um grande número de usuário e manipula grande volume de dados. Desvantagens: tráfego intenso de rede e custo elevado em software e hardware.

Arquitetura Cliente-Servidor – Vantagens: acessíveis, processamento dividido em ambas as máquinas, popular, arquitetura simplifica, pouco tráfego de rede e fácil implementação. Desvantagens: inconsistência de performance.

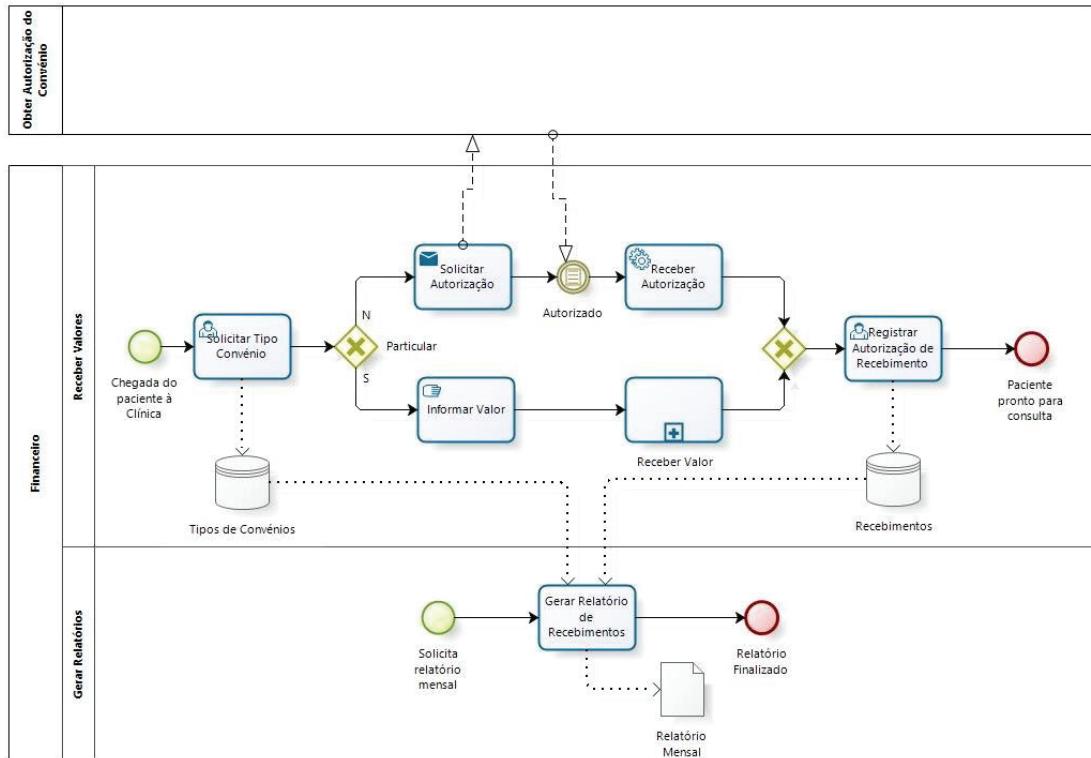
# AULA 2

Arquitetura Distribuída – Vantagens: processamento está distribuído, flexibilidade e desempenho. Desvantagens: elevação na complexidade do desenvolvimento e o equilíbrio entre segurança e performance.

## Atividade 3

A partir da solicitação da Dra. Vânia, modele em BPMN o processo financeiro da Clínica Médica. A Dra. Vânia deseja um sistema que permita controlar o faturamento de sua clínica médica. Ela deseja obter no fim do mês, um relatório com o valor recebido pelas consultas particulares e o faturado pelos convênios. Cada convênio paga um valor diferente por consulta e credita em prazos diferentes (alguns creditam em 45 dias, outros em 60 etc.) a partir da data de apresentação da consulta ao plano. A previsão de pagamento deve aparecer no relatório. O valor da consulta particular é fixo, mas um ou outro paciente pode receber um desconto. Sendo assim, o valor efetivo pago deve ser registrado.

### Expectativa de Resposta:

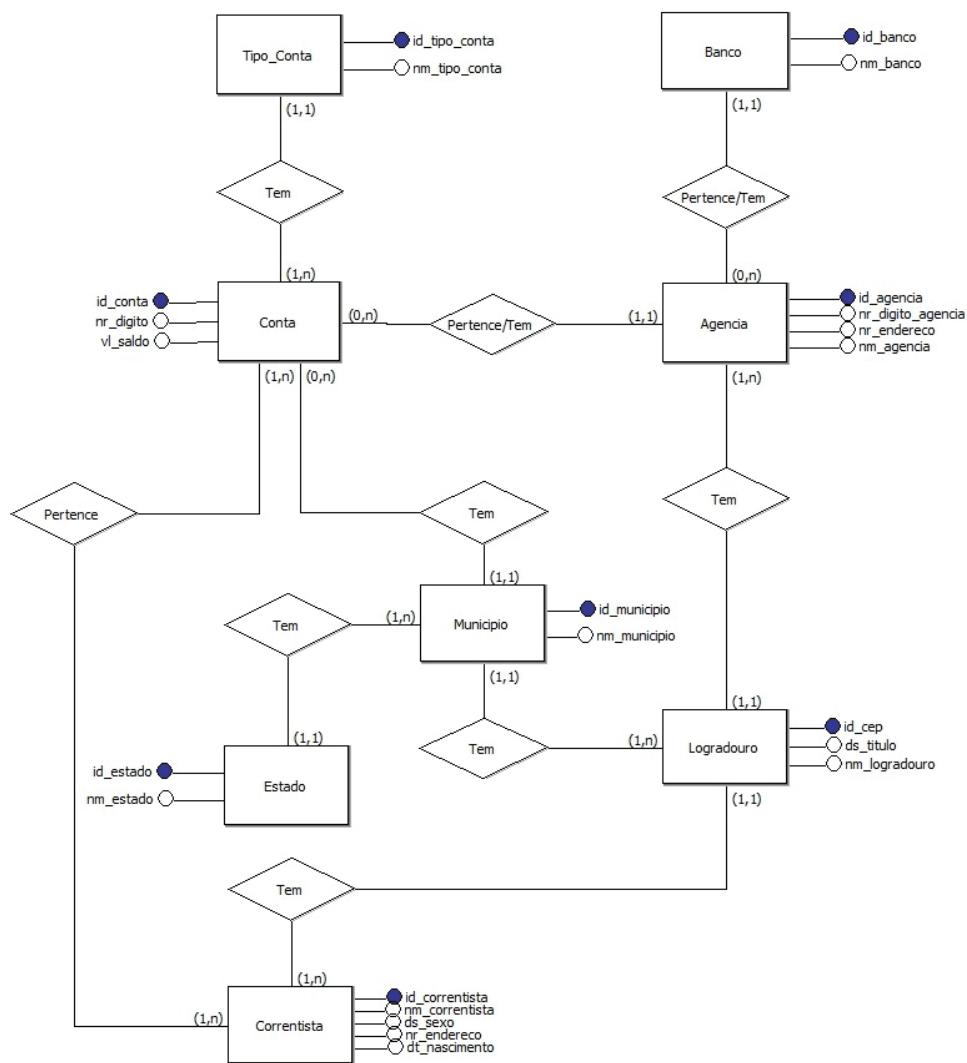


# AULA 3

## Atividade 1

Analise e crie o Modelo Conceitual para um Banco de forma que permita cadastrar nome do banco, código do banco, os dados dos correntistas, como nome do correntista, data de nascimento, sexo, endereço completo do correntista, os dados da conta, tipo do correntista, digito verificador da conta, tipo da conta, valor do saldo, número da agência, nome da agência, endereço completo da agência. No endereço, coloque o nome do logradouro, tipo, título, nome do município, estado e cep.

### Expectativa de Resposta:



# AULA 3

## Atividade 2

Tendo como base o Modelo Conceitual do Banco, converta para o Modelo Lógico Escrito.

Tipo\_Conta(id\_tipo\_conta, nm\_tipo\_conta)

Banco(id\_banco, nm\_banco)

Conta(id\_conta, id\_agencia, id\_tipo\_conta, id\_municipio, nr\_digito, vl\_saldo)

Agencia(id\_agencia, id\_banco)

Municipio(id\_municipio, id\_estado, nm\_municipio)

Logradouro(id\_cep, id\_municipio, nm\_logradouro, ds\_titulo)

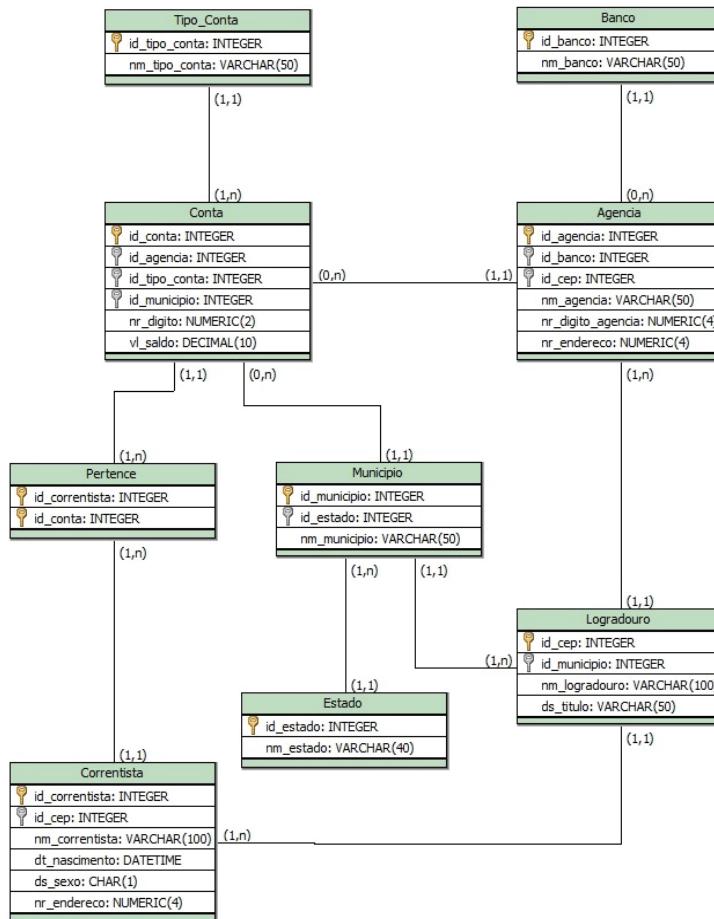
Estado(id\_estado, nm\_estado)

Correntista(id\_correntista, id\_cep, nm\_correntista, dt\_nascimento, ds\_sexo, nr\_endereco)

Pertence(id\_correntista, id\_conta)

Crie os diagramas de Engenharia da Informação do Modelo Lógico Escrito para o Banco.

## Expectativa de Resposta:



# AULA 3

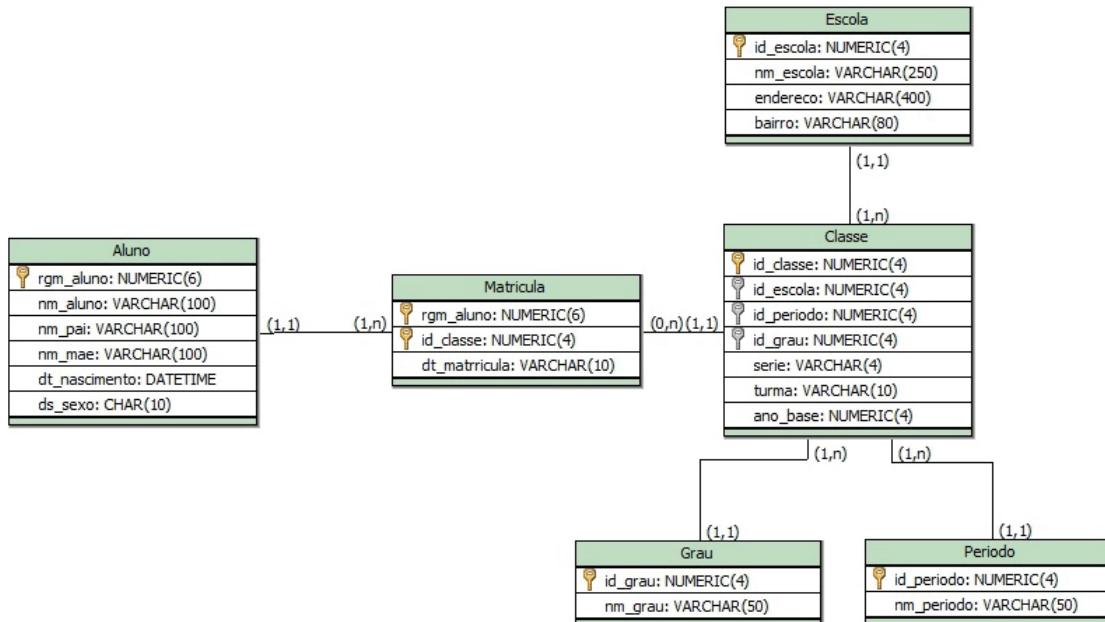
## Atividade 3

Considere o banco de dados com o seguinte esquema

Aluno(rgm\_aluno, nm\_aluno, nm\_pai, nm\_mae, dt\_nascimento, ds\_sexo)  
Matricula(id\_classe, rgm\_aluno, dt\_matricula)  
Classe(id\_classe, ano\_base, serie, turma, id\_escola, id\_grau, id\_periodo)  
Escola(id\_escola, nm\_escola, endereco, bairro)  
Grau(id\_grau, nm\_grau)  
Periodo(id\_periodo, nm\_periodo)

Usando a notação apresentada nesta aula, construa o esquema dia-gramático para o banco de dados com a notação de Engenharia da Informação.

### Expectativa de Resposta:

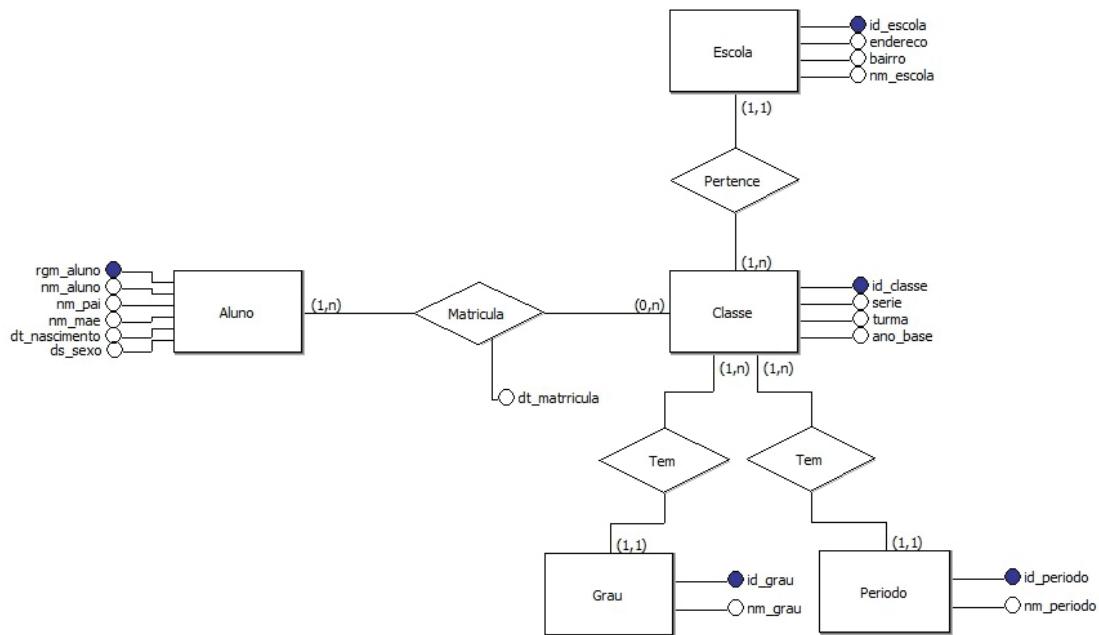


## Atividade 4

Com base na atividade 4, construa o DER.

# AULA 3

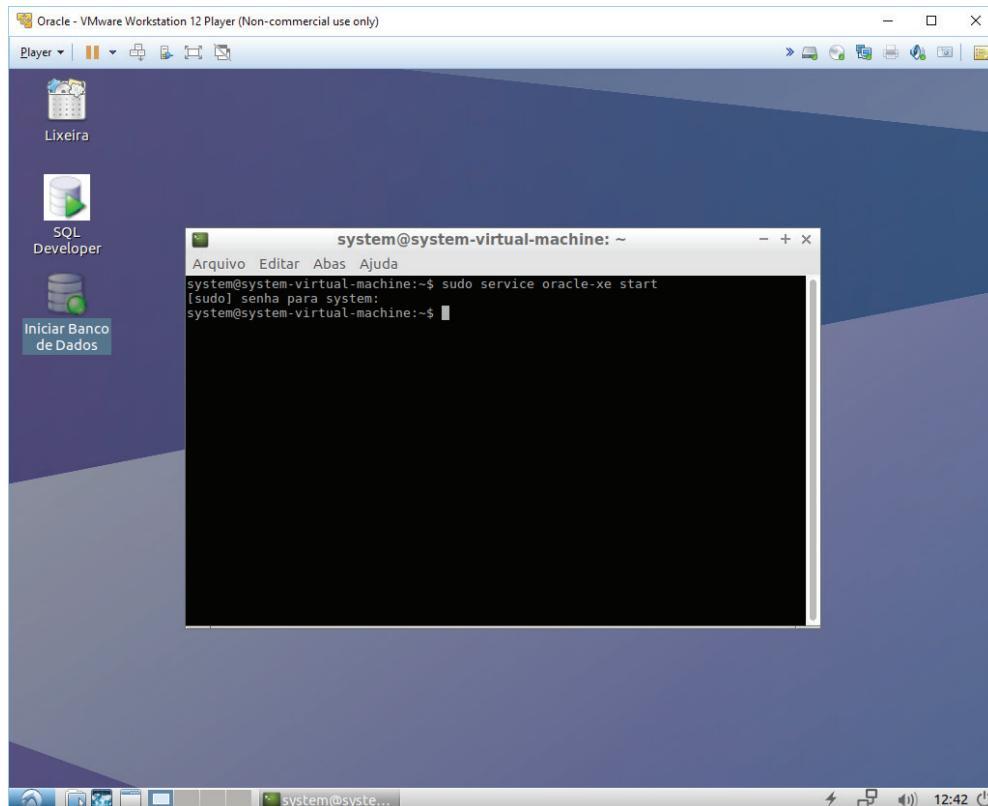
## Expectativa de Resposta:



# AULA 4

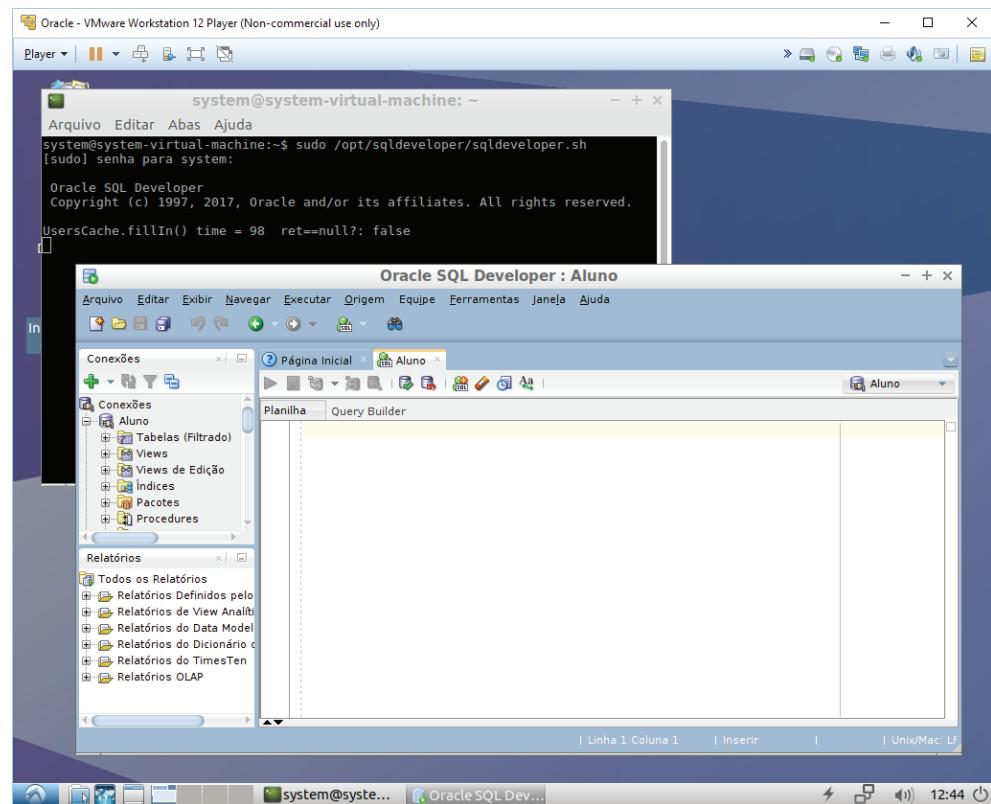
No LXTerminal digite os seguintes comandos e tire os print das telas para comprovar que estão funcionando corretamente.

```
sudo service oracle-xe start
```



```
sudo /opt/sqldeveloper/sqldeveloper.sh
```

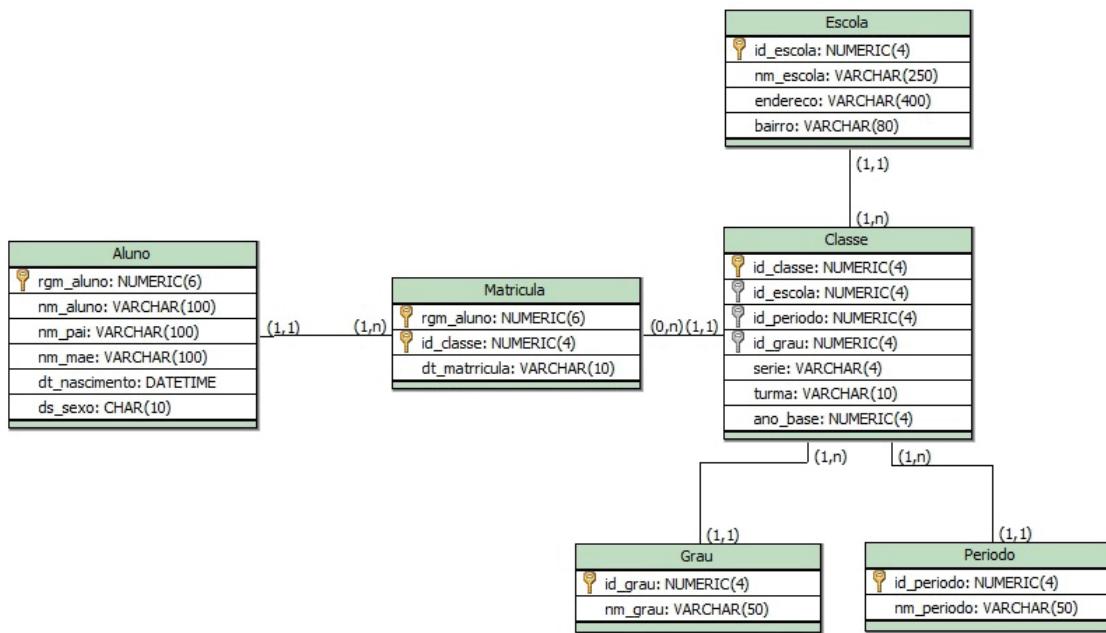
# AULA 4



# AULA 5

## Atividade 1

Com base no esquema diagramático de Engenharia da Informação defina o script SQL



```

-- se existir
DROP TABLE TB_MATRICULA;
DROP TABLE TB_ALUNO;
DROP TABLE TB_CLASSE;
DROP TABLE TB_ESCOLA;
DROP TABLE TB_GRAU;
DROP TABLE TB_PERIODO;

-- criando a tabela de escola
CREATE TABLE TB_ESCOLA(
    id_escola    NUMBER(4),
    nm_escola    VARCHAR2(250) CONSTRAINT nm_escola_nn NOT NULL,
    ds_endereco  VARCHAR2(400) CONSTRAINT end_escola_nn NOT NULL,
    ds_bairro    VARCHAR2(80),
    CONSTRAINT tb_escola_id_escola_pk PRIMARY KEY (id_escola)
);

-- criando a tabela de grau
CREATE TABLE TB_GRAU(
    id_grau      NUMBER(4),
    nm_grau      VARCHAR2(50) CONSTRAINT nm_grau_nn NOT NULL,
    CONSTRAINT tb_grau_id_grau_pk PRIMARY KEY (id_grau)
);
  
```

# AULA 5

```
-- criando a tabela de periodo
CREATE TABLE TB_PERIODO(
    id_periodo  NUMBER(4),
    nm_periodo  VARCHAR2(50) CONSTRAINT nm_periodo_nn NOT NULL,
    CONSTRAINT tb_periodo_id_periodo_pk PRIMARY KEY (id_periodo)
);

-- criando a tabela de classe
CREATE TABLE TB_CLASSE(
    id_classe    NUMBER(4),
    nr_ano_base   NUMBER(4) CONSTRAINT ano_base_classe_nn NOT NULL,
    id_escola    NUMBER(4),
    id_grau       NUMBER(4),
    id_periodo   NUMBER(4),
    ds_serie     VARCHAR2(4) CONSTRAINT serie_classe_nn NOT NULL,
    ds_turma     VARCHAR2(10)CONSTRAINT turma_classe_nn NOT NULL,
    CONSTRAINT tb_classe_id_classe_pk PRIMARY KEY (id_classe),
    CONSTRAINT tb_classe_id_escola_fk FOREIGN KEY(id_escola)
        REFERENCES TB_ESCOLA(id_escola),
    CONSTRAINT tb_classe_id_grau_fk FOREIGN KEY(id_grau)
        REFERENCES TB_GRAU(id_grau),
    CONSTRAINT tb_classe_id_periodo_fk FOREIGN KEY(id_periodo)
        REFERENCES TB_PERIODO(id_periodo),
    CONSTRAINT tb_classe_nr_ano_base_ck CHECK(nr_ano_base>2017)
);

-- criando a tabela de aluno
CREATE TABLE TB_ALUNO(
    rgm_aluno    NUMBER(6),
    nm_nome      VARCHAR2(100)CONSTRAINT nome_aluno_nn NOT NULL,
    nm_pai        VARCHAR2(40) CONSTRAINT pai_aluno_nn NOT NULL,
    nm_mae        VARCHAR2(40) CONSTRAINT mae_aluno_nn NOT NULL,
    dt_nascimento DATE      CONSTRAINT dt_nasc_aluno_nn NOT NULL,
    ds_sexo       CHAR(1)      CONSTRAINT sexo_aluno_nn NOT NULL,
    CONSTRAINT tb_aluno_rgm_aluno_pk PRIMARY KEY (rgm_aluno),
    CONSTRAINT tb_aluno_ds_sexo_ck CHECK (ds_sexo IN ('M','F'))
);

-- criando a tabela de matricula
CREATE TABLE TB_MATRICULA(
    id_classe    NUMBER(4),
    rgm_aluno    NUMBER(6),
    dt_matricula  DATE CONSTRAINT data_matricula_nn NOT NULL,
    CONSTRAINT tb_matricula_pk PRIMARY KEY (id_classe,rgm_aluno),
    CONSTRAINT tb_matricula_rgm_aluno_fk FOREIGN KEY(rgm_aluno)
        REFERENCES TB_ALUNO(rgm_aluno),
    CONSTRAINT tb_matricula_id_classe_fk FOREIGN KEY(id_classe)
        REFERENCES TB_CLASSE(id_classe)
);
```

# AULA 6

## Atividade 1

Com base na tabela gere o script que insere os dados da TB\_EQUIPE

Equipe	
Ferrari	Itália
Force India-Mercedes	Índia
Haas-Ferrari	Estados Unidos
McLaren-Honda	Reino Unido
Mercedes	Alemanha
Red Bull-TAG Heuer	Áustria
Renault	França
Sauber-Ferrari	Suíça
Toro Rosso	Itália
Williams-Mercedes	Reino Unido

## Expectativa de Resposta

```
INSERT INTO TB_PAIS VALUES (15,'Índia',NULL);
INSERT INTO TB_PAIS VALUES (16,'Estados Unidos',NULL);
INSERT INTO TB_PAIS VALUES (17,'México',NULL);
INSERT INTO TB_EQUIPE VALUES (1,'Renault',4);
INSERT INTO TB_EQUIPE VALUES (2,'Ferrari',13);
INSERT INTO TB_EQUIPE VALUES (3,'McLaren-Honda',8);
INSERT INTO TB_EQUIPE VALUES (4,'Williams-Mercedes',8);
INSERT INTO TB_EQUIPE VALUES (5,'Force India-Mercedes',15);
INSERT INTO TB_EQUIPE VALUES (6,'Haas-Ferrari',16);
INSERT INTO TB_EQUIPE VALUES (7,'Mercedes',2);
INSERT INTO TB_EQUIPE VALUES (8,'Red Bull-TAG Heuer',9);
INSERT INTO TB_EQUIPE VALUES (9,'Sauber-Ferrari',11);
INSERT INTO TB_EQUIPE VALUES (10,'Toro Rosso',13);
```

## Atividade 2

Faça uma pesquisa e complete a TB\_PILOTO com os pilotos do calendário de 2017.

## Expectativa de Resposta

```
INSERT INTO TB_PILOTO VALUES (
8,' LANCE STROLL',TO_DATE('29/10/1998','dd/mm/yyyy'),12,'M',4
);
INSERT INTO TB_PILOTO VALUES (
9,' SERGIO PÉREZ',TO_DATE('26/01/1998','dd/mm/yyyy'),17,'M',5
);
```

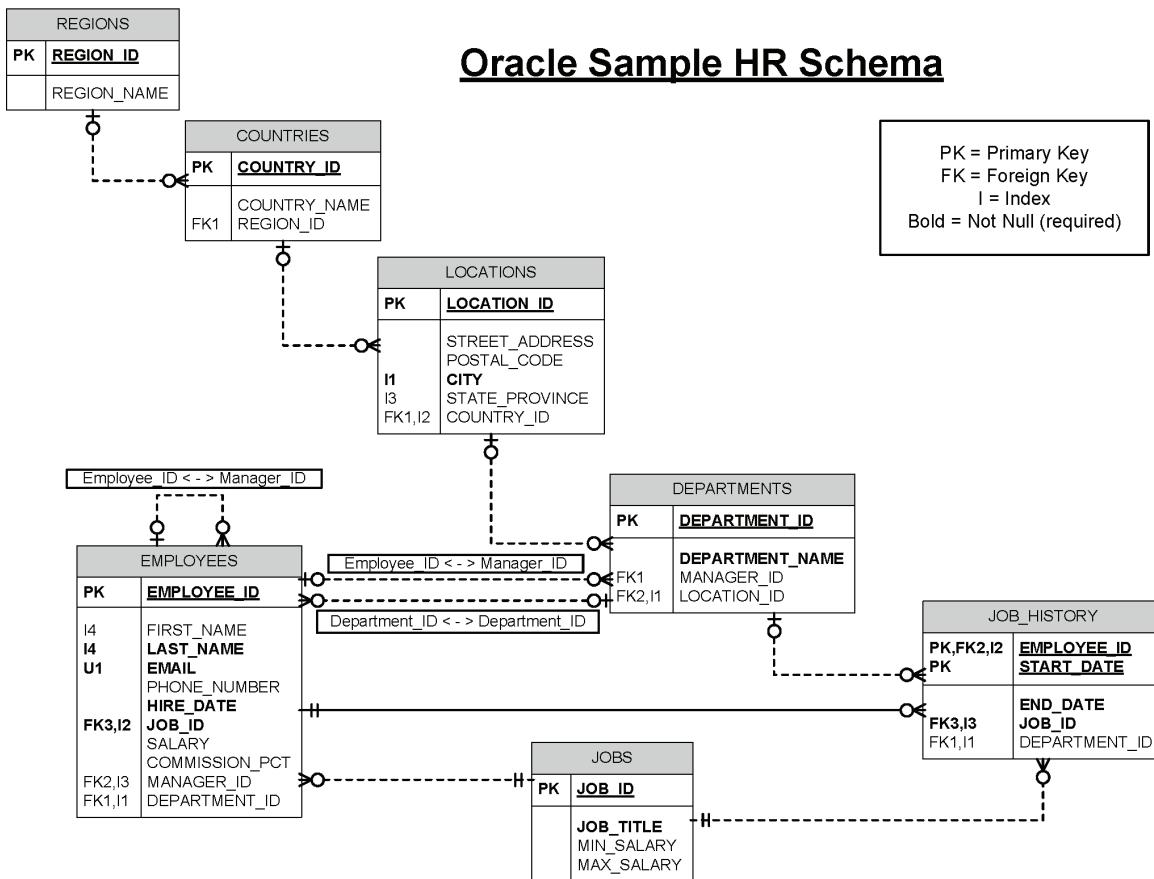
# AULA 6

```
INSERT INTO TB_PILOTO VALUES (
10,' ESTEBAN OCON',TO_DATE('17/09/1996','dd/mm/yyyy'),17,'M',5
);
INSERT INTO TB_PILOTO VALUES (
11,' ROMAIN GROSJEAN',TO_DATE('17/04/1986','dd/mm/yyyy'),4,'M',6
);
INSERT INTO TB_PILOTO VALUES (
12,' KEVIN MAGNUSEN',TO_DATE('05/10/1992','dd/mm/yyyy'),5,'M',6
);
INSERT INTO TB_PILOTO VALUES (
13,' JENSON BUTTON',TO_DATE('19/01/1980','dd/mm/yyyy'),8,'M',3
);
INSERT INTO TB_PILOTO VALUES (
14,' JENSON BUTTON',TO_DATE('19/01/1980','dd/mm/yyyy'),8,'M',3
);
INSERT INTO TB_PILOTO VALUES (
15,' LEWIS HAMILTON',TO_DATE('07/01/1975','dd/mm/yyyy'),8,'M',7
);
INSERT INTO TB_PILOTO VALUES (
16,' VALTTERI BOTTAS',TO_DATE('28/08/1989','dd/mm/yyyy'),3,'M',7
);
INSERT INTO TB_PILOTO VALUES (
17,' DANIEL RICCIARDO',TO_DATE('01/07/1989','dd/mm/yyyy'),8,'M',8
);
INSERT INTO TB_PILOTO VALUES (
18,' MAX VERSTAPPEN',TO_DATE('30/09/1987','dd/mm/yyyy'),6,'M',8
);
INSERT INTO TB_PILOTO VALUES (
19,' MARCUS ERICSSON',TO_DATE('02/09/1990','dd/mm/yyyy'),11,'M',9
);
INSERT INTO TB_PILOTO VALUES (
20,' ANTONIO GIOVINAZZI',TO_DATE('14/12/93','dd/mm/yy'),13,'M',9
);
INSERT INTO TB_PILOTO VALUES (
21,' PASCAL WEHRLEIN',TO_DATE('18/10/1994','dd/mm/yyyy'),2,'M',9
);
INSERT INTO TB_PILOTO VALUES (
22,' DANIIL KVYAT',TO_DATE('26/04/94','dd/mm/yy'),14,'M',10
);
INSERT INTO TB_PILOTO VALUES (
23,' CARLOS SAINZ JR',TO_DATE('01/09/94','dd/mm/yy'),7,'M',10
);
```

# AULA 7

## Atividade 1

Com base no DER, resolva os exercícios:



- a) Mostrar o nome, a data de admissão, cargo e o código do departamento de todos os empregados que tenham o mesmo departamento que o funcionário de nome Steven King. Utilize a tabela EMPLOYEES.

### Expectativa de Resposta:

```

SELECT first_name, hire_date, job_id, department_id
FROM   EMPLOYEES
WHERE  department_id = (SELECT department_id
                        FROM   EMPLOYEES
                        WHERE  first_name = 'Steven'
                        AND   last_name = 'King');
    
```

# AULA 7

b) Mostrar o nome, código do departamento e cargo de todos os empregados que trabalhem nos departamentos que fazem parte da cidade 'Seattle'. Utilize as tabelas: EMPLOYEES, DEPARTMENTS e LOCATIONS.

### Expectativa de Resposta:

```
SELECT first_name, department_id, job_id
FROM EMPLOYEES
WHERE department_id IN (
    SELECT department_id
        FROM DEPARTMENTS
        WHERE location_id = (
            SELECT location_id
                FROM LOCATIONS
                WHERE city = 'Seattle'));
```

c) Exibir código do departamento, nome, cargo de todos os empregados que pertençam ao departamento denominado 'Sales'. Tabelas EMPLOYEES e DEPARTMENTS.

### Expectativa de Resposta:

```
SELECT department_id, first_name, job_id
FROM EMPLOYEES
WHERE department_id = (SELECT department_id
                        FROM DEPARTMENTS
                        WHERE department_name = 'Sales');
```

d) Selecionar nome, cargo e salário de todos os empregados cujo salário seja maior que a média salarial de todos os empregados.

### Expectativa de Resposta:

```
SELECT first_name, job_id, salary
FROM EMPLOYEES
WHERE salary > (SELECT AVG(salary)
                 FROM EMPLOYEES);
```

e) Exibir, nome, salário, data de admissão e código do departamento de todos os empregados que trabalhem no mesmo departamento que o funcionário de nome Steven King e tenham o salário maior que a média salarial dos empregados.

### Expectativa de Resposta:

```
SELECT first_name, salary, hire_date, department_id
```

# AULA 7

```
FROM EMPLOYEES
WHERE department_id IN (SELECT department_id
                         FROM EMPLOYEES
                         WHERE first_name = 'Steven'
                               AND last_name = 'King')
      AND salary > (SELECT AVG(salary)
                      FROM EMPLOYEES);
```

f) Crie uma QUERY que mostre o número do empregado, o último nome e o salário de todos os empregados que ganhem mais que a média salarial. Ordenar o resultado em ordem crescente de salários. Coloque um alias no cabeçalho da cada coluna.

### Expectativa de Resposta:

```
SELECT employee_id AS "Número do Empregado",
       last_name AS «Último Nome»,
       salary AS "Salário"
  FROM EMPLOYEES
 WHERE salary > (SELECT AVG(salary)
                  FROM EMPLOYEES)
 ORDER BY salary;
```

g) Crie uma QUERY que mostre o último nome, o número do departamento e o código do cargo de todos os empregados cuja localização do departamento seja igual a 1700

### Expectativa de Resposta:

```
SELECT last_name AS "Último Nome",
       department_id AS "Código do Dpto",
       job_id AS "Código do Cargo"
  FROM EMPLOYEES
 WHERE department_id IN (SELECT department_id
                           FROM DEPARTMENTS
                           WHERE location_id = 1700);
```

h) Crie uma QUERY que mostre o último nome e o salário de todos os funcionários que tenham o King como gerente.

### Expectativa de Resposta:

```
SELECT last_name, salary
  FROM EMPLOYEES
```

# AULA 7

```
WHERE manager_id = (SELECT employee_id  
                    FROM EMPLOYEES  
                   WHERE first_name = 'Steven'  
                         AND last_name = 'King');
```

- i) Crie uma QUERY que mostre o número do departamento, o último nome e o código do cargo para cada empregado que esteja no departamento 'Executive'

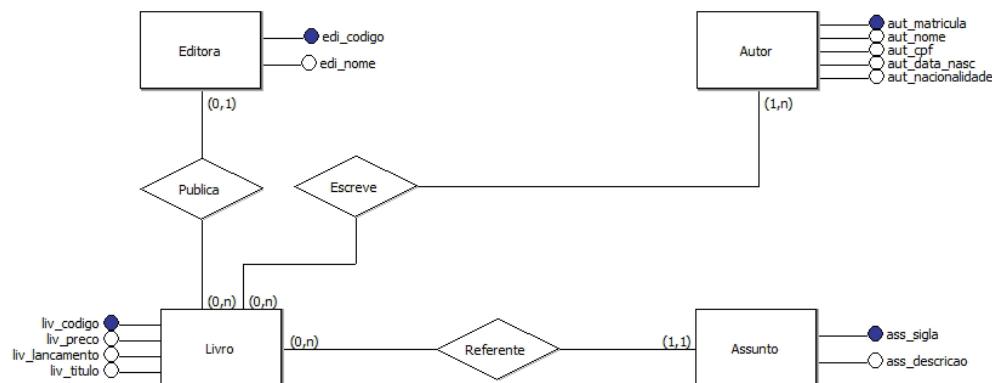
## Expectativa de Resposta:

```
SELECT department_id, last_name, job_id  
FROM EMPLOYEES  
WHERE department_id IN (  
    SELECT department_id  
        FROM DEPARTMENTS  
       WHERE department_name = 'Executive');
```

# AULA 8

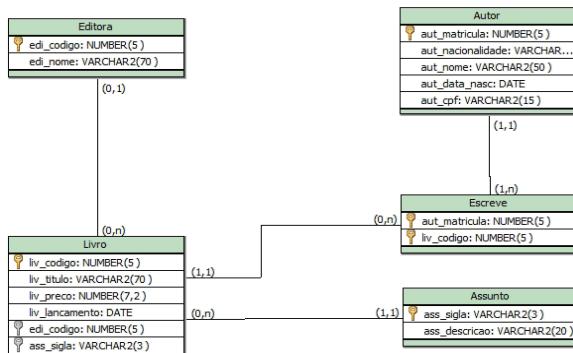
## Atividade 1

Com base no DER, resolva os exercícios:



a) Gere o diagrama lógico do Banco de Dados

## Expectativa de Resposta:



b) Crie o script com os comandos SQL.

## Expectativa de Resposta:

```

CREATE TABLE Editora (
    edi_codigo NUMBER(5),
    edi_nome VARCHAR2(70),
    CONSTRAINT editora_edi_codigo_pk
    PRIMARY KEY (edi_codigo)
);

```

# AULA 8

```
CREATE TABLE Livro (
    liv_codigo NUMBER(5),
    liv_titulo VARCHAR2(70),
    liv_preco NUMBER(7,2),
    liv_lancamento DATE,
    edi_codigo NUMBER(5),
    ass_sigla VARCHAR2(5),
    CONSTRAINT livro_edi_codigo_fk
    FOREIGN KEY(edi_codigo)
    REFERENCES Editora (edi_codigo),
    CONSTRAINT livro_ass_sigla_fk
    FOREIGN KEY (ass_sigla)
    REFERENCES ASSUNTO (ass_sigla),
    CONSTRAINT livro_liv_codigo_pk
    PRIMARY KEY (liv_codigo)
);

CREATE TABLE Autor (
    aut_matricula NUMBER(5),
    aut_nacionalidade VARCHAR2(40),
    aut_nome VARCHAR2(50),
    aut_data_nasc DATE,
    aut_cpf VARCHAR2(15),
    CONSTRAINT autor_aut_matricula_pk
    PRIMARY KEY (aut_matricula)
);

CREATE TABLE Assunto (
    ass_sigla VARCHAR2(5),
    ass_descricao VARCHAR2(200),
    CONSTRAINT assunto_ass_sigla_pk
    PRIMARY KEY (ass_sigla)
);

CREATE TABLE Escreve (
    aut_matricula NUMBER(5),
    liv_codigo NUMBER(5),
    CONSTRAINT escreve_pk
    PRIMARY KEY(aut_matricula,liv_codigo),
    CONSTRAINT escreve_aut_matricula_fk
    FOREIGN KEY(aut_matricula)
    REFERENCES Autor (aut_matricula),
    CONSTRAINT escreve_liv_codigo_fk
    FOREIGN KEY(liv_codigo)
    REFERENCES Livro (liv_codigo)
);
```

# AULA 8

c) Insira os dados apresentados nas tabelas.

LIVRO					
liv_codigo	liv_titulo	liv_preco	liv_lancamento	edi_codigo	ass_sigla
1	banco de dados para web	32,20	10/01/1999	1	BAN
2	programando em linguagem c	30,00	01/10/1997	1	PRO
3	programando em linguagem c++	115,50	01/11/1998	3	PRO
4	banco de dados na bioinformática	48,00		2	BAN
5	redes de computadores	42,00	01/09/1996	2	RED

ESCREVE		ASSUNTO	
liv_codigo	aut_matricula	ass_sigla	ass_descricao
1	1	BAN	Banco de Dados
2	1	PRO	Programação
3	2	RED	Redes
4	3	SIS	Sistemas Operacionais
5	4		

EDITORA		AUTOR	
edi_codigo	edi_nome	aut_matricula	aut_nome
1	Mirandela	1	Luiz
2	Editora Via Norte	2	Hugo
3	Editora Ilhas Tijucas	3	Joaquim
4	Maria José	4	Regina

## Expectativa de Resposta:

```

INSERT INTO EDITORA (edi_codigo, edi_nome) VALUES (1, 'Mirandela');
INSERT INTO EDITORA (edi_codigo, edi_nome) VALUES (2, 'Editora Via Norte');
INSERT INTO EDITORA (edi_codigo, edi_nome) VALUES (3, 'Editora Ilhas Tijucas');
INSERT INTO EDITORA (edi_codigo, edi_nome) VALUES (4, 'Maria José');

-----
INSERT INTO ASSUNTO (ass_sigla, ass_descricao) VALUES ('BAN', 'Banco de Dados');
INSERT INTO ASSUNTO (ass_sigla, ass_descricao) VALUES ('PRO', 'Programação');

```

# AULA 8

```
'Programação');
INSERT INTO ASSUNTO (ass_sigla, ass_descricao) VALUES ('RED',
'Redes');
INSERT INTO ASSUNTO (ass_sigla, ass_descricao) VALUES ('SIS',
'Sistemas Operacionais');
-----
INSERT INTO AUTOR (aut_matricula, aut_nome, aut_cpf, aut_dt-
nasc, aut_nacionalidade) VALUES ('1', 'Luiz', '12345678911',
'15051954', 'Brasileiro');
INSERT INTO AUTOR (aut_matricula, aut_nome, aut_cpf, aut_dt-
nasc, aut_nacionalidade) VALUES ('2', 'Hugo', '14534567895',
'01121975', 'Brasileiro');
INSERT INTO AUTOR (aut_matricula, aut_nome, aut_cpf,
aut_dtnasc, aut_nacionalidade) VALUES ('3', 'Joaquim',
'12543678990', '13021990', 'Brasileiro');
INSERT INTO AUTOR (aut_matricula, aut_nome, aut_cpf,
aut_dtnasc, aut_nacionalidade) VALUES ('4', 'Regiana',
'12365678987', '20101978', 'Brasileiro');
-----
INSERT INTO LIVRO (liv_codigo, liv_titulo, liv_preco, liv_
lancamento, edi_codigo, ass_sigla) VALUES (1, 'Banco de dados
para web', 32.20, '10011999', 1, 'BAN');
INSERT INTO LIVRO (liv_codigo, liv_titulo, liv_preco, liv_
lancamento, edi_codigo, ass_sigla) VALUES (2, 'Programando em
linguagem C', 30.00, '01101997', 1, 'PRO');
INSERT INTO LIVRO (liv_codigo, liv_titulo, liv_preco, liv_
lancamento, edi_codigo, ass_sigla) VALUES (3, 'Programando em
linguagem C++', 115.50, '01111998', 3, 'PRO');
INSERT INTO LIVRO (liv_codigo, liv_titulo, liv_preco, liv_
lancamento, edi_codigo, ass_sigla) VALUES (4, 'Banco de dados
na bioinformática', 48.00, NULL, 2, 'BAN');
INSERT INTO LIVRO (liv_codigo, liv_titulo, liv_preco, liv_
lancamento, edi_codigo, ass_sigla) VALUES (5, 'Redes de com-
putadores', 42.00, '01091996', 2, 'RED');
-----
INSERT INTO ESCREVE (liv_codigo, aut_matricula) VALUES ('1', '1');
INSERT INTO ESCREVE (liv_codigo, aut_matricula) VALUES ('2', '1');
INSERT INTO ESCREVE (liv_codigo, aut_matricula) VALUES ('3', '2');
INSERT INTO ESCREVE (liv_codigo, aut_matricula) VALUES ('4', '3');
INSERT INTO ESCREVE (liv_codigo, aut_matricula) VALUES ('5', '4');
```

- d) Exibir os nomes dos livros, que custam mais de R\$ 100,00 mos-
trando o as iniciais em maiúsculo e o valor.

## Expectativa de Resposta:

```
SELECT 'Livro: ' || InitCap(liv_titulo) || ' ' || liv_preco
"Preço Livro" FROM LIVRO WHERE (liv_preco>100);
```

# AULA 8

f) Mostrar qual é o livro mais caro, mostrando somente o valor.

## Expectativa de Resposta:

```
SELECT MAX(liv_preco) FROM LIVRO;
```

g) Contar quantas editoras existem.

## Expectativa de Resposta:

```
SELECT COUNT(edi_codigo) FROM EDITORA;
```

h) Mostrar o nome do livro e data de lançamento no formato DD/MM/YYYY dos livros da editora com número 1.

## Expectativa de Resposta:

```
SELECT liv_titulo, TO_CHAR(liv_lancamento, 'DD/MM/YYYY') FROM LIVRO WHERE (edi_codigo = 1);
```

i) Mostrar quantos autores existem no banco de dados.

## Expectativa de Resposta:

```
SELECT COUNT(aut_matricula) FROM AUTOR;
```

j) Mostrar os nomes dos livros em maiúsculo.

## Expectativa de Resposta:

```
SELECT UPPER(liv_titulo) FROM LIVRO;
```

h) Montar a seguinte expressão: "O livro XXX tem YYY caracteres", onde XXX será o nome do livro e YYY a quantidade de caracteres que este livro tem.

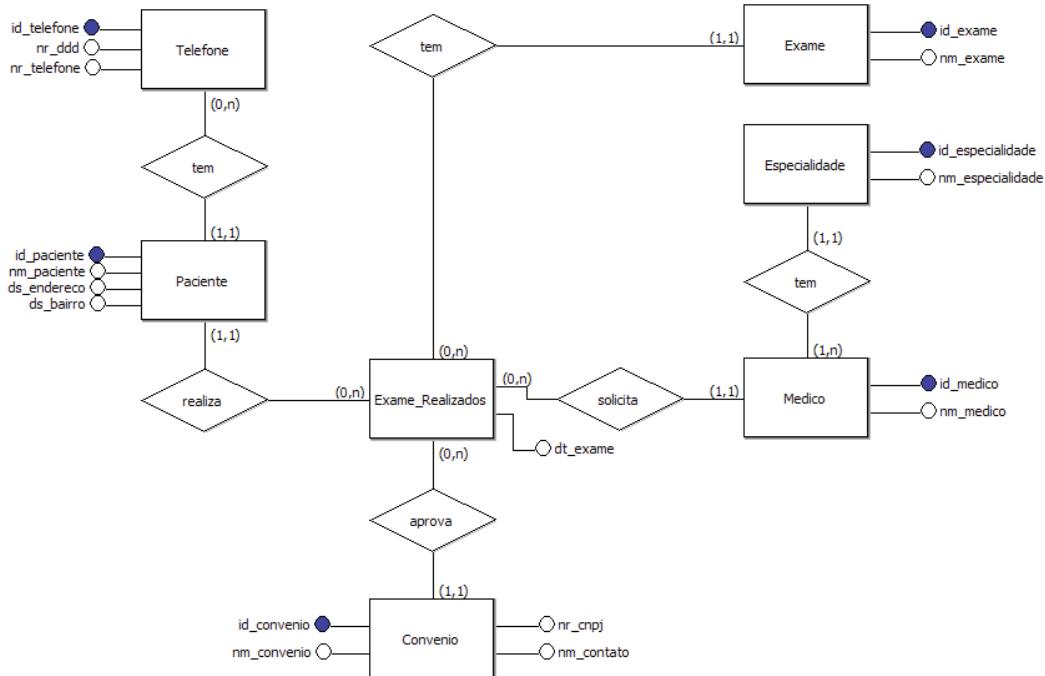
## Expectativa de Resposta:

```
SELECT 'O livro ' || liv_titulo || ' tem ' || length(liv_titulo) || ' caracteres.' FROM LIVRO;
```

# AULA 9

## Atividade 1

Tendo como referência o DER apresentado na figura 16, elabore os comandos SQL solicitados.



**Figura 16:** Diagrama da Clínica Particular Dra. X

**Observe as regras:**

Convênio

id\_convenio – numérico(6) – campo chave

nm\_convenio - varchar2(50) – obrigatório

nr\_cnpj – varchar2(25) – obrigatório

nm\_contato – varchar2(50)

Exame

id\_exame – numérico(6) – campo chave

nm\_exame – varchar2(50) – obrigatório

# AULA 9

Exames\_Realizados

id\_exame\_realizado – num.(6) – chave  
ds\_diagnostico – varchar2(255) – obrigatório  
dt\_exame – data – obrigatório  
id\_medico – numérico(6) – obrigatório  
id\_convenio – numérico(6) – obrigatório  
id\_paciente – numérico(6) – obrigatório  
id\_exame – numérico(6) – obrigatório

Especialidade

id\_especialidade – numérico(6) campo chave  
nm\_especialidade – varchar2(50) – obrigatório

Paciente

id\_paciente – numérico(6) – campo chave  
nm\_paciente – varchar2 (50) – obrigatório  
ds\_endereco – varchar2(50)  
ds\_bairro – varchar2(40)

Telefone

Id\_telefone – numérico(6) – campo chave  
nr\_ddd – numérico(2)  
nr\_telefone – varchar2(8)

Médico

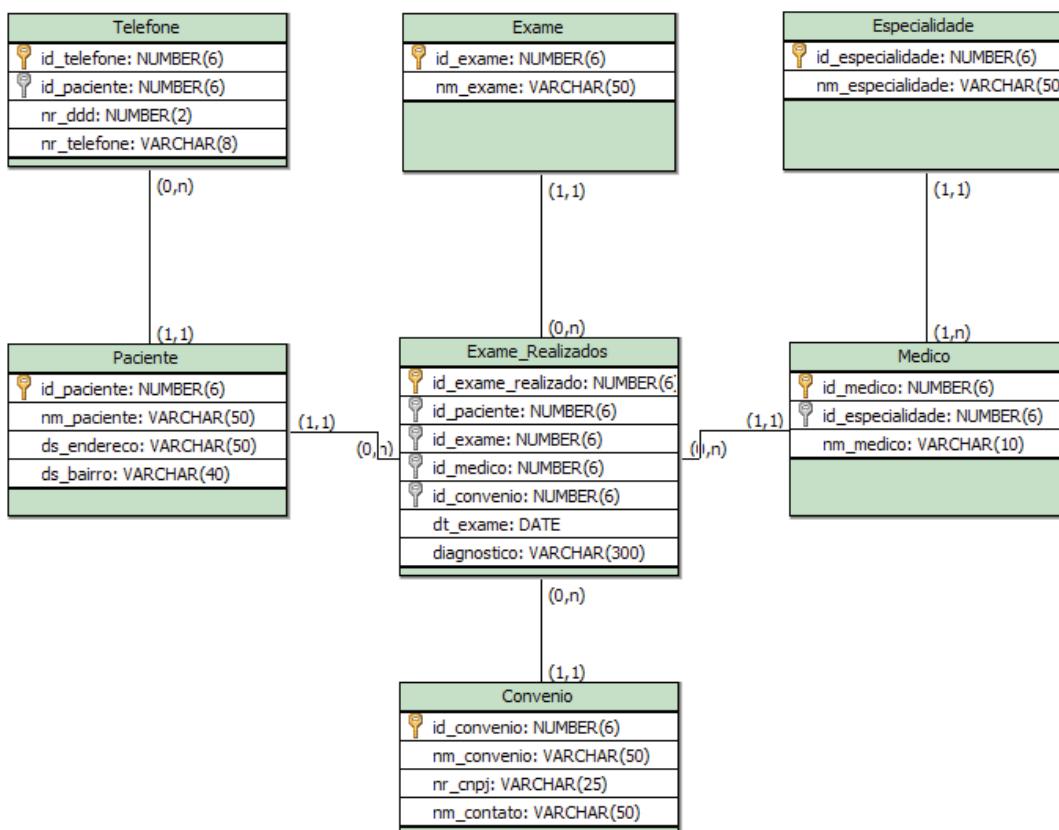
id\_medico – numérico(6) – campo chave  
nm\_medico – varchar2(50) – obrigatório  
id\_especialidade – numérico(6) obrigatório

- Gere o Modelo Lógico e Físico. Criar as tabelas com as restrições apresentada nas observações.

# AULA 9

## Expectativa de Resposta:

Modelo Lógico



Modelo Físico

```

CREATE TABLE Especialidade (
    id_especialidade NUMBER(6),
    nm_especialidade VARCHAR2(50) NOT NULL,
    CONSTRAINT especialidade_pk
    PRIMARY KEY (id_especialidade)
);

CREATE TABLE Telefone (
    id_telefone NUMBER(6),
    id_paciente NUMBER(6),
    nr_ddd NUMBER(2),
    nr_telefone VARCHAR2(8),
    CONSTRAINT telefone_id_telefone_pk
    PRIMARY KEY (id_telefone)
);
    
```

# AULA 9

```
PRIMARY KEY (id_telefone)
);

CREATE TABLE Paciente (
    id_paciente NUMBER(6),
    nm_paciente VARCHAR2(50) NOT NULL,
    ds_endereco VARCHAR2(50),
    ds_bairro VARCHAR(40),
    CONSTRAINT paciente_id_paciente_pk
    PRIMARY KEY (id_paciente)
);

CREATE TABLE Exame (
    id_exame NUMBER(6),
    nm_exame VARCHAR2(50) NOT NULL,
    CONSTRAINT exame_id_exame_pk
    PRIMARY KEY (id_exame)
);

CREATE TABLE Exame_Realizados (
    id_exame_realizados NUMBER(6),
    id_paciente NUMBER(6) NOT NULL,
    id_exame NUMBER(6) NOT NULL,
    id_medico NUMBER(6) NOT NULL,
    id_convenio NUMBER(6) NOT NULL,
    diagnostico VARCHAR2(300) NOT NULL,
    dt_exame DATE NOT NULL,
    CONSTRAINT exame_realizados_pk
    PRIMARY KEY (id_exame_realizados),
    CONSTRAINT exame_real_id_paciente_fk
    FOREIGN KEY(id_paciente) REFERENCES Paciente (id_paciente),
    CONSTRAINT exame_real_id_exame_fk
    FOREIGN KEY(id_exame) REFERENCES Exame (id_exame)
);

CREATE TABLE Medico (
    id_medico NUMBER(6),
    id_especialidade NUMBER(6) NOT NULL,
    nm_medico VARCHAR2(50) NOT NULL,
    CONSTRAINT medico_id_medico_pk
    PRIMARY KEY (id_medico),
    CONSTRAINT medico_id_especialidade_fk
    FOREIGN KEY(id_especialidade)
    REFERENCES Especialidade (id_especialidade)
);

CREATE TABLE Convenio (
    id_convenio NUMBER(6),
```

# AULA 9

```
nm_convenio VARCHAR2(50) NOT NULL,  
nr_cnpj VARCHAR2(25) NOT NULL,  
nm_contato VARCHAR2(50),  
CONSTRAINT convenio_id_convenio_pk  
PRIMARY KEY (id_convenio)  
) ;  
  
ALTER TABLE Telefone  
ADD (  
CONSTRAINT telefone_id_paciente_fk  
FOREIGN KEY(id_paciente)  
REFERENCES Paciente (id_paciente)  
) ;  
  
ALTER TABLE Exame_Realizados  
ADD (  
CONSTRAINT exame_realizados_id_medico_fk  
FOREIGN KEY(id_medico)  
REFERENCES Medico (id_medico)  
) ;  
  
ALTER TABLE Exame_Realizados  
ADD (  
CONSTRAINT exame_real_id_convenio_fk  
FOREIGN KEY(id_convenio)  
REFERENCES Convenio (id_convenio)  
) ;
```

b) Insira 10 registros em cada tabela.

## Expectativa de Resposta:

```
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('1','Amil','1','Maria Paula');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('2','Unimed Paulistana','2','João Carlos');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('3','SulAmérica','3','Carlos Alexandre');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('4','One Health','4','Pedro Almeida');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('5','Bradesco Saúde','5','Giovani Santos');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_CONTATO) VALUES ('6','Intermédica / NotreDame','6','Rogerio Alencar');  
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO, NR_CNPJ, NM_
```

# AULA 9

```
CONTATO) VALUES ('7','Corpore Administradora de Benefi-
cios','7','Luciana Pereira');
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO,NR_CNPJ,NM_CON-
TATO) VALUES ('8','Lincx','8','Aline Vieira');
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO,NR_CNPJ,NM_CON-
TATO) VALUES ('9','Qualicorp Soluções em Saúde','9','Antonio
dos Santos');
INSERT INTO CONVENIO (ID_CONVENIO,NM_CONVENIO,NR_CNPJ,NM_CON-
TATO) VALUES ('10','All Care Beneficios','10','José Antunes');

INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('1','Obstetra');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('2','Oncopediatra');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('3','Pediatria');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('4','Pneumologia');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('5','Neurologia');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('6','Neurocirurgia');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('7','Urologia');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('8','Reumatologia');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('9','Radioterapeuta');
INSERT INTO ESPECIALIDADE (ID_ESPECIALIDADE,NM_ESPECIALIDADE)
VALUES ('10','Patologia');

INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('1', '1', 'João da Silva Sauro');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('2', '1', 'Oswaldo Cruz');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('3', '2', 'Hippocrates');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('4', '3', 'Carlos Chagas');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('5', '4', 'Patch Adams');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('6', '5', 'Roberto M. Rey Jr.');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('7', '1', 'Drauzio Varella');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('8', '6', 'Ivo Pitanguy');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
```

# AULA 9

```
VALUES ('9', '7', 'Jose Mengele');
INSERT INTO MEDICO (ID_MEDICO, ID_ESPECIALIDADE, NM_MEDICO)
VALUES ('10', '8', 'Che Guevara');

INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('1', 'Philip J. Fry', 'Futurama', 'Vila
Futuro');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('2', 'Patrick Estrela', 'Fenda do Biqui-
ni', 'Vila Fenda do Biquini');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('3', 'Tibercio dos Santos Sauro', 'Fenda
do Biquini', 'Vila Fenda do Biquini');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('4', 'Sr. Burns', 'Mansão Burns', 'Vila
dos Milionarios');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('5', 'Pikachu', 'Rua das Alamedas, 21',
'Pokémon');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('6', 'Homer Simpson', 'Evergreen Terrace,
742', 'Springfield');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('7', 'Bart Simpson', 'Evergreen Terrace,
742', 'Springfield');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('8', 'Marge Simpson', 'Evergreen Terrace,
742', 'Springfield');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('9', 'Lisa Simpson', 'Evergreen Terrace,
742', 'Springfield');
INSERT INTO PACIENTE (ID_PACIENTE, NM_PACIENTE, DS_ENDERECO,
DS_BAIRRO) VALUES ('10', 'Maggie Simpson', 'Evergreen Terra-
ce, 742', 'Springfield');

INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('1', '1', '11', '91234567');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('2', '1', '11', '43211234');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('3', '2', '11', '97659123');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('4', '3', '11', '92233445');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('5', '4', '11', '93344556');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('6', '5', '11', '44321234');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
```

# AULA 9

```
TELEFONE) VALUES ('7', '6', '11', '42131232');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('8', '7', '11', '32178769');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('9', '8', '21', '76585324');
INSERT INTO TELEFONE (ID_TELEFONE, ID_PACIENTE, NR_DDD, NR_
TELEFONE) VALUES ('10', '9', '11', '32533232');

INSERT INTO EXAME
VALUES ('1','Próstata');
INSERT INTO EXAME
VALUES ('2','Tomografia da Córnea');
INSERT INTO EXAME
VALUES ('3','Hemograma');
INSERT INTO EXAME
VALUES ('4','Velocidade de Hemossedimentação');
INSERT INTO EXAME
VALUES ('5','Eletrocardiograma');
INSERT INTO EXAME
VALUES ('6','Eletroencefalograma');
INSERT INTO EXAME
VALUES ('7','Uranálise');
INSERT INTO EXAME
VALUES ('8','Audiometria');
INSERT INTO EXAME
VALUES ('9','Paquimetria');
INSERT INTO EXAME
VALUES ('10','Urofluxometria');

INSERT INTO EXAME_REALIZADOS
VALUES ('1','1','2','1','6','Não foi possível obter resultados conclusivos será necessário realizar novamente o exame', to_date('24/11/2017', 'DD/MM/YYYY'));
INSERT INTO EXAME_REALIZADOS
VALUES ('2','3','2','6','1','Foi observado sub formação no cerebelo (hipoplastia) e baixa quantidade de fibras córtex cerebrais no cérebro como grande parte dos exames realizados em outras pessoas com Transtorno do espectro autista', to_date('16/11/2017', 'DD/MM/YYYY'));
INSERT INTO EXAME_REALIZADOS
VALUES ('3','10','1','1','6','Não foi possível obter resultados conclusivos será necessário realizar novamente o exame', to_date('11/09/2017', 'DD/MM/YYYY'));
INSERT INTO EXAME_REALIZADOS
VALUES ('4','5','2','5','4','Nenhuma anormalidade notada no exame, a paciente se mantém com boa saúde cardíaca', to_date('25/12/2016', 'DD/MM/YYYY'));
INSERT INTO EXAME_REALIZADOS
```

# AULA 9

```
VALUES ('5','6','7','1','6','Não foi possível obter resultados conclusivos será necessário realizar novamente o exame', to_date('25/10/2017', 'DD/MM/YYYY'));  
INSERT INTO EXAME_REALIZADOS  
VALUES ('6','3','5','2','7','Deformação e fragilidade da córnea indicam Ceratocone', to_date('10/12/2017', 'DD/MM/YYYY'));  
INSERT INTO EXAME_REALIZADOS  
VALUES ('7','9','10','9','7','Córnea saudável e bem rigida', to_date('09/12/2017', 'DD/MM/YYYY'));  
INSERT INTO EXAME_REALIZADOS  
VALUES ('8','2','6','6','1','Nenhuma anormalidade a reportar pelo exame', '25/09/2017');  
INSERT INTO EXAME_REALIZADOS  
VALUES ('9','8','1','6','1','Nenhuma anormalidade a reportar pelo exame', '31/10/2016');  
INSERT INTO EXAME_REALIZADOS  
VALUES ('10','4','3','3','2','Nenhuma anomalia que remeta a alguma infecção', '10/12/2017');
```

- c) Selecionar os médicos que tem especialidade OBSTETRA e que tenham sobrenome Silva no nome

### Expectativa de Resposta:

```
SELECT nm_medico  
FROM medico  
JOIN especialidade  
ON medico.id_especialidade = especialidade.id_especialidade  
WHERE nm_especialidade='Obstetra'  
AND nm_medico LIKE '%Silva%';
```

- d) Exibir a quantidade de médicos existentes por especialidade, mostrando o nome da especialidade e a quantidade.

### Expectativa de Resposta:

```
SELECT nm_especialidade, count(*)  
FROM especialidade  
JOIN medico  
ON especialidade.id_especialidade = medico.id_especialidade  
GROUP BY nm_especialidade;
```

- e) Exibir o nome do exame, data do exame dos convênios da AMIL

# AULA 9

## Expectativa de Resposta:

```
SELECT nm_exame, dt_exame
FROM exame_realizados
JOIN exame
ON exame.id_exame = exame_realizados.id_exame
JOIN convenio
ON convenio.id_convenio = exame_realizados.id_convenio
WHERE nm_convenio='Amil';
```

- f) Exibir todos os exames (Nome) que foram realizados pelo paciente TIBERCIO DOS SANTOS SAURO no mês de dezembro de 2017.

## Expectativa de Resposta:

```
SELECT nm_exame
FROM exame
JOIN exame_realizados
ON exame.id_exame = exame_realizados.id_exame
JOIN paciente
ON paciente.id_paciente = exame_realizados.id_paciente
WHERE nm_paciente = 'Tibercio dos Santos Sauro';
```

- g) Exibir todos os médicos que tem a mesma especialidade do médico JOÃO DA SILVA SAURO

## Expectativa de Resposta:

```
SELECT nm_medico
FROM medico
WHERE id_especialidade IN (
SELECT id_especialidade
FROM medico
WHERE nm_medico = 'João da Silva Sauro'
);
```

- h) Contar quantos exames foram realizados por Convênio, somente convênios que tem mais de 1 exames (Mostrar o nome do convênio e a quantidade).

## Expectativa de Resposta:

```
SELECT nm_convenio, count(*)
```

# AULA 9

```
FROM exame_realizados
JOIN convenio
ON convenio.id_convenio = exame_realizados.id_convenio
GROUP BY nm_convenio
HAVING count(*) > 1
```

i) Contar quantos pacientes fizeram Exames por convenio.

## Expectativa de Resposta:

```
SELECT nm_convenio convenio, count(*) pacientes
FROM exame_realizados er
JOIN convenio co
ON co.id_convenio = er.id_convenio
GROUP BY nm_convenio;
```

j) Exibir nome do paciente, nome do convênio, nome do médico, e nome do exame dos exames realizados em dezembro de 2017.

## Expectativa de Resposta:

```
SELECT nm_paciente, nm_convenio, nm_medico, nm_exame, dt_exame
FROM exame_realizados
JOIN exame
ON exame.id_exame = exame_realizados.id_exame
JOIN medico
ON medico.id_medico = exame_realizados.id_medico
JOIN convenio
ON convenio.id_convenio = exame_realizados.id_convenio
JOIN paciente
ON paciente.id_paciente = exame_realizados.id_paciente
WHERE dt_exame BETWEEN DO_DATE('01/12/2017', 'dd/mm/yyyy')
AND DO_DATE('31/12/2017', 'dd/mm/yyyy');
```

k) Exibir a quantidade de exames que são realizados por médico, ignorando o médico JOAO DA SILVA SAURO. Mostrar o nome do médico e a quantidade).

## Expectativa de Resposta:

```
SELECT nm_medico "Nome do Médico", count(*) "Quantidade de
Exames"
FROM exame ex
JOIN exame_realizados er
```

# AULA 9

```
ON ex.id_exame = er.id_exame
JOIN medico me
ON me.id_medico = er.id_medico
WHERE nm_medico NOT IN ('João da Silva Sauro')
GROUP BY nm_medico;
```

- l) Exibir a quantidade de exames que são realizados por mês, mostrando o nome do mês e o total de exames.

## Expectativa de Resposta:

```
SELECT to_char(dt_exame, 'MONTH') "Mês",
       count(*) "Total Exames"
FROM exame_realizados
GROUP BY to_char(dt_exame, 'MONTH');
```

- m) Exibir todos os pacientes que realizaram o exame de próstata.

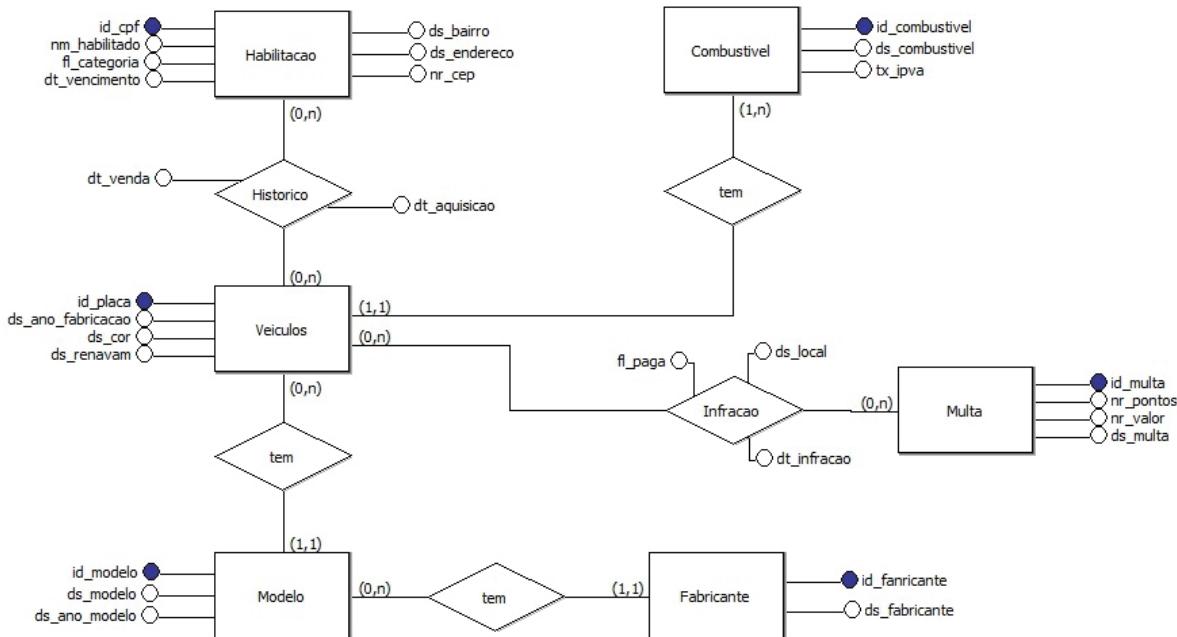
## Expectativa de Resposta:

```
SELECT nm_paciente
FROM paciente
JOIN exame_realizados
ON paciente.id_paciente = exame_realizados.id_paciente
JOIN exame
ON exame.id_exame = exame_realizados.id_exame
WHERE nm_exame = 'Próstata';
```

# AULA 9

## Atividade 2

Tendo como referência o DER apresentado na figura a seguir, elabore os comandos SQL solicitados.



**Figura 17:** Diagrama Base do Departamento de Trânsito

### Habilitacao

id\_cpf – varchar2(11) – campo chave  
 nm\_habilitado - varchar2(50) – obrigatório  
 ds\_endereco – varchar2(50) – obrigatório  
 nr\_cep – varchar2(8) – obrigatório  
 ds\_bairro – varchar2(40) – obrigatório  
 fl\_categoria – char(1) – obrigatório  
 dt\_vencimento – date – obrigatório

### Historico

id\_cpf – varchar2(11) – campo chave  
 id\_placa – varchar2(8) – campo chave

# AULA 9

dt\_aquisicao – date – obrigatório  
dt\_venda – date

Veículos

id\_placa – varchar2(8) – campo chave  
ds\_renavam – varchar2(13) – obrigatório  
ds\_ano\_fabric – numérico(4) – obrigatório  
ds\_cor – varchar2(40) – obrigatório  
id\_combustivel - numérico (2) – obrigatório  
id\_fabricante – numérico(4) – obrigatório

Modelo

id\_modelo – numérico(4) – campo chave  
ds\_modelo – varchar2(40) – obrigatório  
ds\_ano\_modelo – numérico(4) – obrigatório  
id\_fabricante – numérico(4) – obrigatório

Fabricante

id\_fabricante – numérico(4) – campo chave  
ds\_fabricante – varchar2(40) – obrigatório

Infração

id\_multa – numérico(6) – campo chave  
id\_placa – varchar2(8) – campo chave  
dt\_infracao – date – obrigatório  
ds\_local – varchar2(50) – obrigatório  
fl\_paga – varchar2(1) – aceitando somente S ou N

Multa

id\_multa – numérico(6) – campo chave  
ds\_multa – varchar2 (50) – obrigatório  
nrPontos – numérico(2) – obrigatório  
nr\_valor – numérico(12,2) - obrigatório

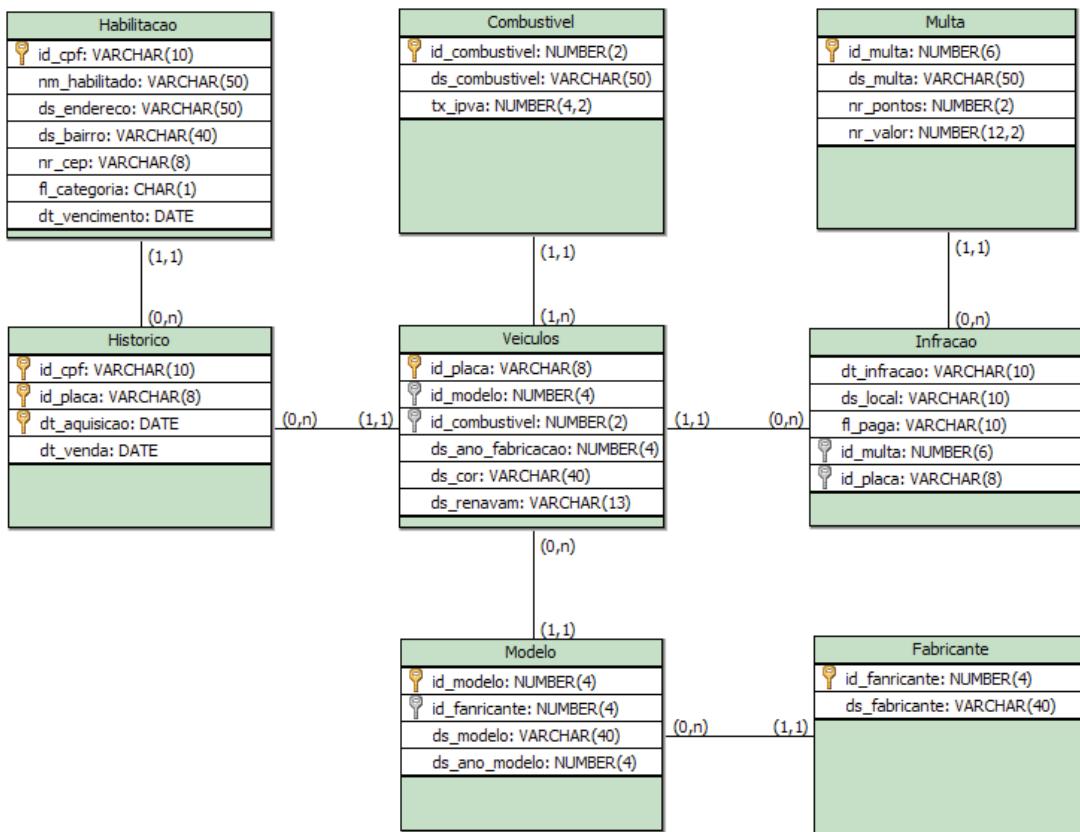
# AULA 9

Combustível

id\_combustivel - numérico(2) – campo chave  
 ds\_combustivel – varchar2 (50) – obrigatório  
 tx\_ipva – numérico(4,2) – obrigatório

- a) Criar as tabelas com os relacionamentos e regras de CONSTRAINT. Insira 2 proprietários de veículos e 7 infrações de transito distribuída entre os veículos.

## Expectativa de Resposta:



Modelo Físico

```

CREATE TABLE Habilitacao (
    id_cpf VARCHAR2(10),
    nm_habilitado VARCHAR2(50) NOT NULL,
    ds_endereco VARCHAR2(50) NOT NULL,
  
```

# AULA 9

```
ds_bairro VARCHAR2(40) NOT NULL,  
nr_cep VARCHAR2(8) NOT NULL,  
fl_categoria CHAR(1) NOT NULL,  
dt_vencimento DATE NOT NULL,  
CONSTRAINT habilitacao_id_cpf_pk  
PRIMARY KEY (id_cpf)  
);  
  
CREATE TABLE Fabricante (  
id_fabricante NUMBER(4),  
ds_fabricante VARCHAR2(40) NOT NULL,  
CONSTRAINT fabricante_id_fabricante_pk  
PRIMARY KEY (id_fabricante)  
);  
  
CREATE TABLE Combustivel (  
id_combustivel NUMBER(2),  
ds_combustivel VARCHAR2(50) NOT NULL,  
tx_ipva NUMBER(4,2) NOT NULL,  
CONSTRAINT combustivel_id_combustivel_pk  
PRIMARY KEY (id_combustivel)  
);  
  
CREATE TABLE Veiculos (  
id_placa VARCHAR2(8),  
id_modelo NUMBER(4) NOT NULL,  
id_combustivel NUMBER(2) NOT NULL,  
ds_ano_fabricacao NUMBER(4) NOT NULL,  
ds_cor VARCHAR2(40) NOT NULL,  
ds_renavam VARCHAR2(13) NOT NULL,  
CONSTRAINT veiculos_id_placa_pk  
PRIMARY KEY (id_placa),  
CONSTRAINT veiculos_id_combustivel_fk  
FOREIGN KEY(id_combustivel)  
REFERENCES Combustivel (id_combustivel)  
);  
  
CREATE TABLE Modelo (  
id_modelo NUMBER(4),  
id_fabricante NUMBER(4) NOT NULL,  
ds_modelo VARCHAR2(40) NOT NULL,  
ds_ano_modelo NUMBER(4) NOT NULL,  
CONSTRAINT modelo_id_modelo_pk  
PRIMARY KEY (id_modelo),  
CONSTRAINT modelo_id_fabricante_fk  
FOREIGN KEY(id_fabricante)  
REFERENCES Fabricante (id_fabricante)  
);
```

# AULA 9

```
CREATE TABLE Multa (
    id_multa NUMBER(6),
    ds_multa VARCHAR2(50) NOT NULL,
    nr_pontos NUMBER(2) NOT NULL,
    nr_valor NUMBER(12,2) NOT NULL,
    CONSTRAINT multa_id_multa_pk
    PRIMARY KEY (id_multa)
);

CREATE TABLE Historico (
    id_cpf VARCHAR2(10),
    id_placa VARCHAR2(8),
    dt_aquisicao DATE NOT NULL,
    dt_venda DATE,
    CONSTRAINT historico_pk
    PRIMARY KEY(id_cpf,id_placa,dt_aquisicao),
    CONSTRAINT historico_id_cpf_fk
    FOREIGN KEY(id_cpf)
    REFERENCES Habilitacao (id_cpf),
    CONSTRAINT historico_id_placa_fk
    FOREIGN KEY(id_placa)
    REFERENCES Veiculos (id_placa)
);

CREATE TABLE Infracao (
    dt_infracao DATE,
    ds_local VARCHAR2(100) NOT NULL,
    fl_paga CHAR(1),
    id_multa NUMBER(6),
    id_placa VARCHAR2(8),
    CONSTRAINT infracao_id_multa_fk
    FOREIGN KEY(id_multa)
    REFERENCES Multa (id_multa),
    CONSTRAINT infracao_id_placa_fk
    FOREIGN KEY(id_placa)
    REFERENCES Veiculos (id_placa),
    CONSTRAINT infracao_pk
    PRIMARY KEY(id_multa,id_placa,dt_infracao),
    CONSTRAINT inf_paga_ck CHECK (fl_paga = 'S' OR fl_paga = 'N')
);

ALTER TABLE Veiculos
ADD (
    CONSTRAINT veiculos_id_modelo_fk
    FOREIGN KEY(id_modelo)
    REFERENCES Modelo (id_modelo)
);
```

# AULA 9

```
INSERT INTO habilitacao(id_cpf, nm_habilitado, ds_endereco,
nr_cep, ds_bairro, fl_categoria, dt_vencimento)
VALUES (1112223334, 'Paulo Ricardo', 'Avenida Industrial',
09080501, 'Campestre', 'B', '02/10/2022');
INSERT INTO habilitacao(id_cpf, nm_habilitado, ds_endereco,
nr_cep, ds_bairro, fl_categoria, dt_vencimento)
VALUES (1817902658, 'Patrick Henrique', 'Rua Carlão',
05030301, 'Utinga', 'E', '02/10/2019');
INSERT INTO habilitacao(id_cpf, nm_habilitado, ds_endereco,
nr_cep, ds_bairro, fl_categoria, dt_vencimento)
VALUES (3309191087, 'Joao da Silva Sauro', 'Rua Doutor Mazei-
des', 07033056, 'Utinga', 'E', '02/10/2019');

INSERT INTO combustivel
VALUES (1, 'GASOLINA', 50.00);
INSERT INTO combustivel
VALUES (2, 'DIESEL', 90.00);

INSERT INTO fabricante (id_fabricante, ds_fabricante)
VALUES ('1','Ford');
INSERT INTO fabricante (id_fabricante, ds_fabricante)
VALUES ('2','Chevrolet');
INSERT INTO fabricante (id_fabricante, ds_fabricante)
VALUES ('3','Fiat');

INSERT INTO modelo (id_modelo, id_fabricante, ds_modelo, ds_
ano_modelo)
VALUES ('1','1','Ranger','2009');
INSERT INTO modelo (id_modelo, id_fabricante, ds_modelo, ds_
ano_modelo)
VALUES ('2','2','Spin','2011');
INSERT INTO modelo (id_modelo, id_fabricante, ds_modelo, ds_
ano_modelo)
VALUES ('3','3','Palio','2006');

INSERT INTO veiculos(id_placa, ds_renavam, id_modelo, ds_ano_
fabricacao, ds_cor, id_combustivel)
VALUES ('BIB-1234', '3015022053644', '1', 2008, 'Vermelho',
'1');
INSERT INTO veiculos(id_placa, ds_renavam, id_modelo, ds_ano_
fabricacao, ds_cor, id_combustivel)
VALUES ('ddd-7788', '4987632184756', '2', 2010, 'Preto',
'2');
INSERT INTO veiculos(id_placa, ds_renavam, id_modelo, ds_ano_
fabricacao, ds_cor, id_combustivel)
VALUES ('DRJ-7865', '33091820909', '3', 2005, 'Cinza', '1');
```

# AULA 9

```
INSERT INTO multa
VALUES (1, 'Excesso de limite de velocidade', 3, 150.00);
INSERT INTO multa
VALUES (2, 'Excesso de limite de velocidade', 3, 120.00);
INSERT INTO multa
VALUES (3, 'Não sinalizou ao cruzar a rua', 3, 50.00);
INSERT INTO multa
VALUES (4, 'Não sinalizou ao cruzar a rua', 3, 1200.00);
INSERT INTO multa
VALUES (5, 'Não deu preferência de passagem', 2, 1200.00);
INSERT INTO multa
VALUES (6, 'Não deu preferência de passagem', 4, 1200.00);
INSERT INTO multa
VALUES (7, 'Baixa velocidade em uma via de 80km/h', 1,
1200.00);
INSERT INTO multa
VALUES (8, 'Alta velocidade numa pista de 60km/h', 2,
120.00);
INSERT INTO multa
VALUES (9, 'Alta velocidade numa pista de 60km/h', 2,
120.00);

INSERT INTO histórico (id_placa, id_cpf, dt_aquisicao, dt_
venda)
VALUES ('ddd-7788',1112223334, TO_DATE('01/02/2010','DD-MM-
YYYY'),TO_DATE('07/03/2015','DD-MM-YYYY'));
INSERT INTO histórico (id_placa, id_cpf, dt_aquisicao, dt_
venda)
VALUES ('BIB-1234',1817902658, TO_DATE('05/04/2010','DD-MM-
YYYY'),TO_DATE('08/05/2015','DD-MM-YYYY'));
INSERT INTO histórico (id_placa, id_cpf, dt_aquisicao, dt_
venda)
VALUES ('DRJ-7865',3309191087, TO_DATE('14/01/2010','DD-MM-
YYYY'),TO_DATE('02/03/2015','DD-MM-YYYY'));

INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('BIB-1234',1, '02/12/2016', 'Rua Jabaquara', 'N');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('ddd-7788',2, '05/06/2016', 'Avenida João Ramalho',
'S');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('BIB-1234',3, '22/08/2016', 'Rua Benedito', 'S');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('BIB-1234',4, '23/06/2016', 'Rua General Ouro Fino',
```

# AULA 9

```
'S');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('ddd-7788',5, '05/12/2016', 'Avenida dos Estados',
'N');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('BIB-1234',6, '17/12/2016', 'Avenida João Ramalho',
'N');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('ddd-7788',7, '30/09/2016', 'Rua Estrada Velha',
'S');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('DRJ-7865',8, '30/08/2016','Rua Estrada Nova', 'S');
INSERT INTO infracao (id_placa, id_multa, dt_infracao, ds_lo-
cal, fl_paga)
VALUES ('DRJ-7865',9, '30/08/2016','Avenida Braz Leme, 508',
'S');
```

2b) Selecionar os veículos (Placa) que tem combustível GASOLINA e que foram fabricados em 2008.

## Expectativa de Resposta:

```
SELECT id_placa placa
FROM veiculos ve
JOIN combustivel co
ON ve.id_combustivel = co.id_combustivel
WHERE UPPER(ds_combustivel) = 'GASOLINA'
AND ds_ano_fabricao = 2008;
```

c) Exibir a quantidade de condutores existentes por categoria, mostrando a categoria e o total de condutores.

## Expectativa de Resposta:

```
SELECT fl_categoria Categoria, count(*) "Quantidade de condu-
tores"
FROM habilitacao
GROUP BY fl_categoria;
```

# AULA 9

- d) Exibir o nome do proprietário, data de aquisição e data da venda do veículo com placa “BIB-1234”

## Expectativa de Resposta:

```
SELECT nm_habilitado "Proprietário",
       hi.dt_aquisicao "Data de aquisição",
       hi.dt_venda "Data de venda"
  FROM habilitacao ha
 JOIN historico hi
    ON ha.id_cpf = hi.id_cpf
 WHERE UPPER(hi.id_placa) = 'BIB-1234';
```

- e) Exibir todas as infrações (Nome da Infração, Valor e Data da Infração) que foram cometidas pelo veículo “BIB-1234” no ano de 2016.

## Expectativa de Resposta:

```
SELECT mu.ds_multa "Infrações",
       mu.nr_valor "Valor em R$",
       fr.dt_infracao "Data da Infração"
  FROM multa mu
 JOIN infracao fr
    ON mu.id_multa = fr.id_multa
 WHERE UPPER(fr.id_placa) = 'BIB-1234'
   AND to_char(fr.dt_infracao, 'YYYY') = '2016';
```

- f) Exibir todos os veículos que tem o mesmo combustível do veículo DDD-7788

## Expectativa de Resposta:

```
SELECT id_placa "Placa"
  FROM veiculos ve
 JOIN combustivel co
    ON ve.id_combustivel = co.id_combustivel
 WHERE co.ds_combustivel = (
    SELECT ds_combustivel
      FROM veiculos ve
     JOIN combustivel co
```

# AULA 9

```
        ON ve.id_combustivel = co.id_
combustivel
        WHERE UPPER(ve.id_placa) =
'DDD-7788'
) ;
```

- g) Contar quantos veículos existem por tipo de combustível, sómente dos combustíveis que tem mais de 100 ocorrências. (Mostrar o nome do combustível e a quantidade).

## Expectativa de Resposta:

```
SELECT co.ds_combustivel "Combustivel", count(*) "Quantidade"
FROM veiculos ve
JOIN combustivel co
ON ve.id_combustivel = co.id_combustivel
GROUP BY co.ds_combustivel
HAVING count(*) > 100;
```

- h) Exibir nome do proprietário, Placa do veículo, marca, modelo, ano de fabricação, combustível dos veículos adquiridos pelo CPF 11122233344

## Expectativa de Resposta:

```
SELECT ha.nm_habilitado "Proprietario",
ve.id_placa "Placa",
fa.ds_fabricante "Marca",
mo.ds_modelo "Modelo",
ve.ds_ano_fabricacao "Ano Fabricação",
co.ds_combustivel "Combustivel"
FROM habilitacao ha
JOIN historico hi
ON ha.id_cpf = hi.id_cpf
JOIN veiculos ve
ON ve.id_placa = hi.id_placa
JOIN combustivel co
ON ve.id_combustivel = co.id_combustivel
JOIN modelo mo
ON mo.id_modelo = ve.id_modelo
JOIN fabricante fa
ON fa.id_fabricante = mo.id_fabricante
```

# AULA 9

```
WHERE hi.id_cpf = '1112223334';
```

- i) Exibir a quantidade de infrações por placa, ignorando o veículo XXX-9988. (Mostrar a placa, nome da infração e a quantidade).

## Expectativa de Resposta:

```
SELECT hi.id_placa "Placa",
       mu.ds_multa "Infração",
       count(*) "Quantidade"
  FROM historico hi
 JOIN veiculos ve
    ON hi.id_placa = ve.id_placa
 JOIN infracao fr
    ON fr.id_placa = ve.id_placa
 JOIN multa mu
    ON mu.id_multa = fr.id_multa
 GROUP BY hi.id_placa, mu.ds_multa
 HAVING hi.id_placa <> 'XXX-9988';
```

- j) Exibir a quantidade de infrações aplicadas por mês, mostrando o nome do mês e o total de infrações.

## Expectativa de Resposta:

```
SELECT to_char(dt_infracao, 'MM') "Mês",
       count(*) "Quantidade de Infrações"
  FROM infracao
 GROUP BY to_char(dt_infracao, 'MM');
```

- k) Criar uma visão, chamada visão01 que mostre o nome do proprietário, placa do veículo, data da infração, local da infração, descrição da multa e valor da multa cujo carro seja do condutor JOAO DA SILVA SAURO.

## Expectativa de Resposta:

```
CREATE OR REPLACE VIEW visao1 AS
SELECT ha.nm_habilitado "Proprietário",
       hi.id_placa "Placa",
       fr.dt_infracao "Data",
       fr.ds_local "Local",
       mu.ds_multa "Infração",
       mu.nr_valor "Valor em R$"
```

# AULA 9

```
FROM historico hi
JOIN veiculos ve
ON hi.id_placa = ve.id_placa
JOIN infracao fr
ON fr.id_placa = ve.id_placa
JOIN multa mu
ON mu.id_multa = fr.id_multa
JOIN habilitacao ha
ON ha.id_cpf = hi.id_cpf
WHERE UPPER(ha.nm_habilitado) = 'JOAO DA SILVA SAURO';
```

l) Exibir o total de multas a receber do ano 2016

## Expectativa de Resposta:

```
SELECT to_char(dt_infracao, 'YYYY') "Ano",
       count(*) "Quantidade de multas"
  FROM infracao
 GROUP BY to_char(dt_infracao, 'YYYY')
 HAVING to_char(dt_infracao, 'YYYY') = '2016';
```

m) Calcular o valor total (taxa IPVA + multas) a pagar para o veículo  
BIB-1234

## Expectativa de Resposta:

```
SELECT avg(co.tx_ipva) + sum(mu.nr_valor) "Valor Total"
  FROM combustivel co
 JOIN veiculos ve
ON co.id_combustivel = ve.id_combustivel
 JOIN infracao fr
ON fr.id_placa = ve.id_placa
 JOIN multa mu
ON fr.id_multa = mu.id_multa
 WHERE UPPER(ve.id_placa) = 'BIB-1234';
```

n) Calcular os pontos na Habilitação do condutor JOAO DA SILVA  
SAURO

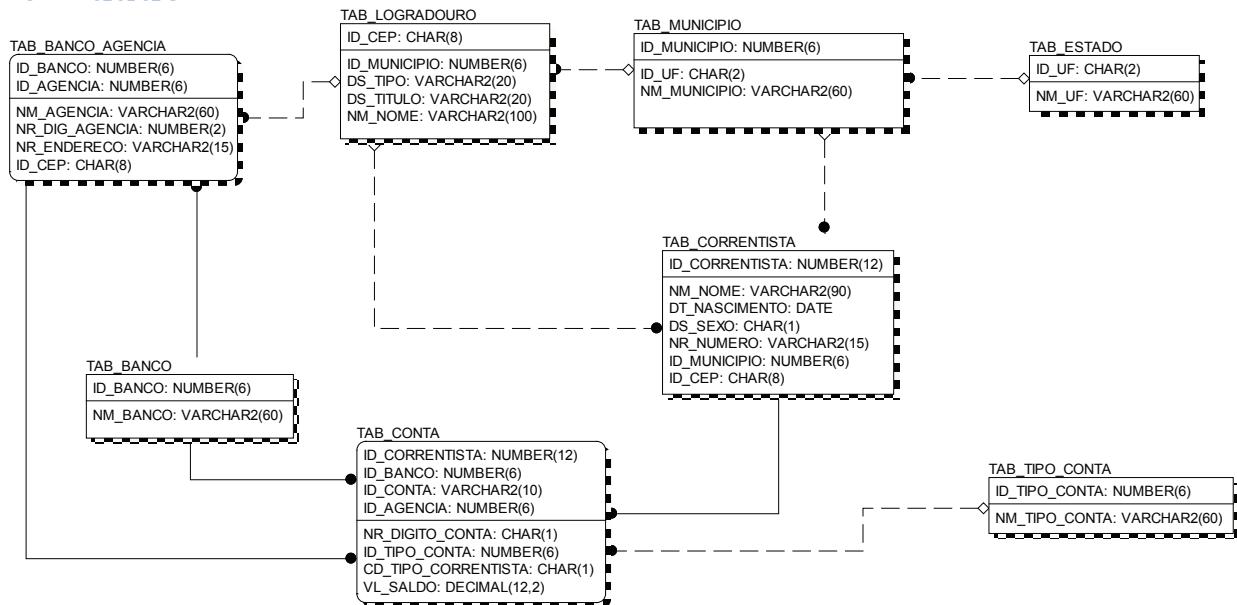
# AULA 9

## Expectativa de Resposta:

```
SELECT sum(mu.nr_pontos) "Pontos"
FROM historico hi
JOIN veiculos ve
ON hi.id_placa = ve.id_placa
JOIN infracao fr
ON fr.id_placa = ve.id_placa
JOIN multa mu
ON mu.id_multa = fr.id_multa
JOIN habilitacao ha
ON ha.id_cpf = hi.id_cpf
WHERE UPPER(ha.nm_habilitado) = 'JOAO DA SILVA SAURO';
```

# AULA 10

## Atividade 1



- a). Crie um PROCEDURE PL/SQL para incluir dados na tabela tab\_correntista.

### Expectativa de Resposta:

```

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE inserir_correntista (
    v_nome IN VARCHAR2,
    v_data IN DATE,
    v_sexo IN CHAR,
    v_numero IN VARCHAR2,
    v_id_municipio IN NUMBER,
    v_id_cep IN CHAR
) IS
    v_correntista NUMBER;
BEGIN
    SELECT MAX(id_correntista)+1 INTO v_correntista
    FROM tab_correntista;

    INSERT INTO tab_correntista
    VALUES (v_correntista, v_nome, v_data, upper(v_sexo), v_numero,
            v_id_municipio, v_id_cep);
    DBMS_OUTPUT.PUT_LINE
    
```

# AULA 10

```
('Correntista inserido com sucesso id:' || v_correntista);
END;

EXEC inserir_correntista('Maria Machado', '10/10/10', 'F',
'320', 1, '17400010');
```

- b) Faça um bloco PL/SQL para excluir um dado, onde o usuário digita o código do correntista a ser excluído.

## Expectativa de Resposta:

```
SET SERVEROUTPUT ON;
ACCEPT correntista NUMBER FORMAT '999' PROMPT 'Digite o código do correntista que deseja excluir';
BEGIN
    DELETE FROM tab_correntista WHERE id_correntista = &correntista;
END;
```

- c) 3. Faça um bloco PL/SQL para que atualize o saldo de um determinado cliente. O saldo e o cliente deverão ser digitados pelo usuário.

## Expectativa de Resposta:

```
SET SERVEROUTPUT ON;
ACCEPT v_id_correntista NUMBER FORMAT '999' PROMPT 'Digite o código do correntista';
ACCEPT v_saldo NUMBER FORMAT '999,999.99' PROMPT 'Digite o saldo';
BEGIN
    UPDATE tab_conta
    SET vl_saldo = &v_saldo
    WHERE id_correntista = &v_id_correntista;
    DBMS_OUTPUT.PUT_LINE ('Conta alterada com sucesso!');
END;
```

## ■ REFERÊNCIAS BIBLIOGRÁFICAS

- ASCENCIO, A. F. G.; ARAÚJO, G. S.; Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++; São Paulo; Pearson Prentice Hall, 2010.
- CARVALHO, V.; MySQL - Comece com o principal banco de dados open source do mercado; Editora: Casa Código
- DATE, C. J. Introdução a Sistemas de Bancos de Dados. São Paulo, Campus, 2004.
- DATE, C. J.; SQL e Teoria Relacional; Editora: Novatec; 1º Ed.;
- ELMASRI, R.; NAVATHE, S. B.; Sistema de banco de dados.; 6. ed.; São Paulo; Pearson Addison Wesley, 2011.
- GONÇALVES, E.; SQL – Uma abordagem para banco de dados Oracle; Editora: Casa Código
- GREENWALD, R.; STACKOWIAK R.; STERN J.; Oracle Essentials: Oracle Database 11g; Fourth Edition; O'Reilly Media, 2008.
- HEUSER, C. A.; Projeto de banco de dados. 6.ed.; Porto Alegre; Bookman, 2009.
- OLIVEIRA, N. G.; Modelagem de Dados. São Paulo, Grande Porte Treinamentos em Informática, 2016.
- ORACLE; Database PL/SQL Language Reference; disponível em: <https://docs.oracle.com/cloud/latest/db112/LNPLS/toc.htm>, acessado em: 22/01/2018;
- RAMARKRISHNAN, R.; Sistemas de gerenciamento de banco de dados.; 3.ed. Porto Alegre; AMGH, 2001.
- ROB, P.; CORONEL, C.; Sistemas de Banco de Dados: projeto, Implementação e gerenciamento; revisão técnica Ana Paula Appel; 8º Ed.; São Paulo: Cengage Learning, 2014.
- TECHONTHENET; Oracle / PLSQL; disponível em: <https://www.techonthenet.com/oracle/>, acessado em: 22/01/2018;
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J.; Estruturas de dados usando C.; São Paulo; Pearson Makron Books, 2010.
- TEOREY, T. J.; Projeto e Modelagem de Banco de Dados; Editora: Elsevier; 1º Ed.; 2013.
- ZIVIANI, N.; Projeto de algoritmos: com implementação em Java e C++; consultoria em Java e C++ de Fabiano Cupertino Botelho; São Paulo: Cengage Learning, 2015.

