06

Recuperando dados usando o comando SELECT

Recuperando dados usando o comando SELECT

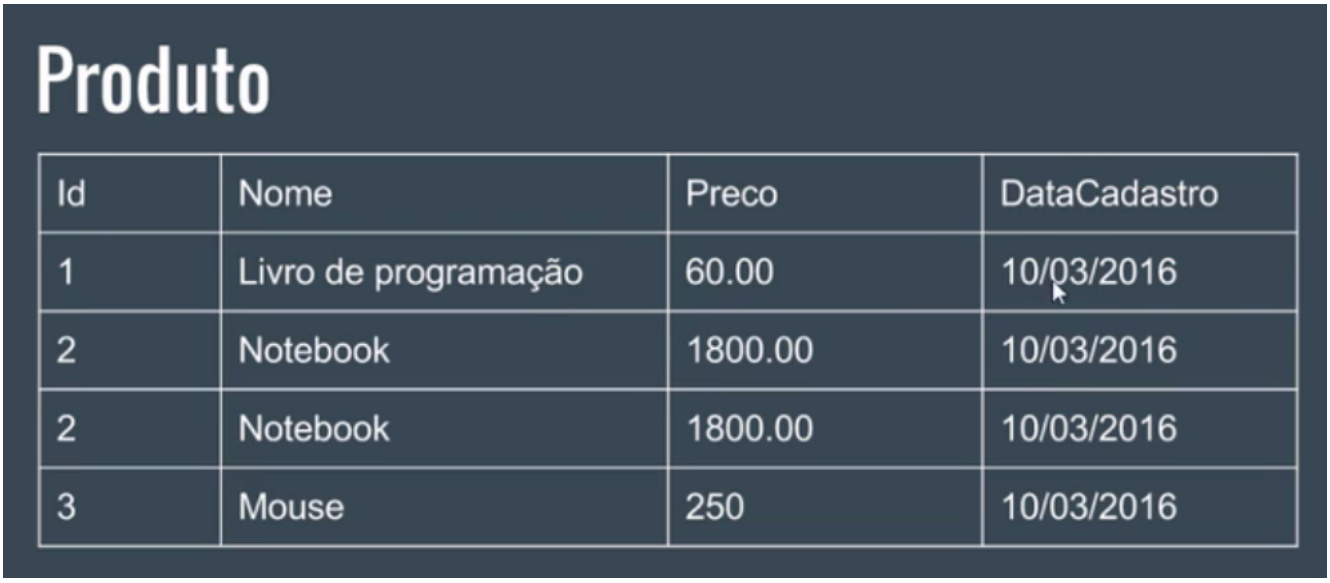
Bem-vindos ao segundo curso da Certificação Oracle. No primeiro curso, discutimos bastante sobre como o Oracle funciona, quais são as arquiteturas disponíveis, e principalmente como organizar os nossos dados. Também vimos o que é um banco de dados relacional, quais são os tipos de relacionamentos, além de outros aspectos.

Neste curso, começaremos a trabalhar efetivamente com o Oracle. Nós já mostramos como salvar os nossos dados, montamos o nosso modelo lógico, e agora, iremos criar o modelo físico.

Anteriormente, nós modelamos um sistema semelhante ao de um supermecado. Nós tinhamos um produto e queríamos fazer uma compra, que era realizada por um usuário. De todo este sistema, uma das entidades mais importantes era `Produto`. É por ela que iramos começar!

Definimos que a entidade `Produto` seria composto por um `Id`, `Nome`, `Preco` e `DataCadastro`.

Só que para conseguirmos exemplificar melhor quais são so tipos de dados, iremos usar uma tabela de exemplo com quatro produtos cadastrados.



Temos os seguintes produtos, com seus respectivos preços: um livro de programação (R$ 60,00), dois Notebooks (R$ 1800,00) e um mouse (R$ 250). Todos foram cadastrados no dia 10 de março de 2016.

Agora, que trabalhamos com o modelo físico, precisamos pensar no tipo de informações que queremos salvar. Na primeira coluna da tabela, referente ao `Id` , o tipo será `numero` . O identificador será um número inteiro, porque temos "1, 2 e 3". O `nome` será um texto, com no máximo 200 carateres. O `preco` será semelhante ao `id` , porém, terá casas decimais. Podemos ter um produto que custa R$ 60,30. A última coluna terá um formato de data. Iremos pegar a entidade `Produto` e implementá-la no Oracle. Quando implementarmos tanto o **modelo lógico** com o **modelo físico**, toda entidade se tornará uma tabela.

Vou me conectar no Oracle, vou executar um atalho para abrir o Oracle com o comando `service.msc` . Com o `cmd` iremos abrir o prompt de comando. Com ele já aberto, digitarei `cd desktop` para poder mudar de pasta e vou entrar no Oracle através do SQL *Plus.

```
C:\Users\Alura>cd Desktop

C:\Users\Alura>Desktop>sqlplus
```

Depois, ele irá solicitar um usuário. Vamos usar o usuário criado no primeiro curso (`alura`).

```
Informe o nome do usuario: alura
Informe a senha:
Horario do ultimo log-in bem-sucedido: Qua Abr 13 2016 03:24:17 -03:00
```

Temos agora que criar um tabela para a entidade. Para isto, usaremos o comando `create table` , seguido pelo nome da entidade `Produto` . Porém, uma entidade tem apenas um nome? Ela tem diversos atributos, que irão se transformar em colunas. Então, vamos especificar que o `id` é um número (`number`) que poderá ter até 10 dígitos.

```
SQL> create table Produto (
  2 Id number(10)
```

Precisamos também pensar sobre as regras do `id` . Observe que temos dois produtos com o `id 2` . O objetivo de termos um id é exatamente identificarmos um registro. Mas, se quisermos criar uma promoção com desconto no valor do notebook que

custa R$ 1800,00. Como eu saberei qual é o notebook que me refiro na tabela? Então, o *id´*, além de ser um número, precisa ser único. Lembre-se que este campo tem um nome especial: **chave primária**. No Oracle, para afirmar que se trata de uma chave primária, irei adicionar o comando `primary key`. Com ele, não poderá mais ocorrer o caso em que dois itens tenham o mesmo `id`. No nosso caso, teríamos um `id` do 1 ao 4.

Além de inserir o `primary key`, quero adicionar a coluna `nome`, que será do tipo texto. Para isto, usaremos o `char`, com o qual salvaremos um sequência de caracteres. A quantidade máxima será de 250 caracteres. Em alguns casos, como a palavra `mouse` só iremos utilizar 5 letras. Porém, com o `char`, ficará reservado um espaço de 250 letras - na verdade, o termo apropriado é 250 **bytes**. Com o *enconding* que estamos usando, cada caracter equivale a um byte. A questão é que, em alguns casos, nós não iremos usar os 250 bytes. Para resolver isto, iremos utilizar o `varchar2`, que irá na verdade reservar **até** 250 bytes. E quando o meu registro usar apenas 5 bytes, ele não irá ocupar o restante do espaço.

```
SQL> create table Produto (
  2 Id number(10) primary key,
  3 Nome varchar2(250),
  4 Preco number
```

Nós também teremos uma coluna para o `Preco`, que será de números. O primeiro parâmetro que precisaremos passar nesta linha é a quantidade de dígitos. No nosso caso, iremos especificar que usaremos até 10 dígitos. Porém, desta vez, iremos aceitar d2 casas decimais, o que significa que só aceitaremos até 8 números inteiros.

```
SQL> create table Produto (
  2 Id number(10) primary key,
  3 Nome varchar2(250),
  4 Preco number(10,2));
```

O primeiro parâmetro se refere a quantidade de dígitos totais - incluindo as casa decimais. Logo, teremos oito digitos livres para os números inteiros. Ao fecharmos com o `;` (ponto e vírgula), ele irá imprimir `Tabela criada`.

Como faço para verificar se a minha tabela foi realmente criada? Podemos usar o comando `select`, o nome da minha tabela (`table_name`) do meu `User_tables`. Ele irá mostrar a tabela `Produto`.

```
SQL> select table_name from User_tables;

TABLE_NAME
--------------------------------------------------------------------------------
PRODUTO

SQL>
```

Se quisermos ver a estrutura da tabela, poderemos usar o comando `desc`, juntamente, com o nome da tabela `produto`.

```
SQL> desc produto;

NOME                                                  nulo?     Tipo
----------------------------------------------------- -------- ------------------------------------
 ID                                                     NOT NULL NUMBER(10)
 NOME                                                            VARCHAR2(250)
 PRECO                                                           NUMBER(10,2)
```

```
SQL>
```

Agora, conseguimos ver toda a estrutura da tabela. Todos os tipos de dados que definimos para os campos, se é númerico, um varchar, iremos discutir de forma mais aprofundada em um próximo curso. O importante é conhecermos as pequenas diferenças no dados.

Observe que esquecemos de adicionar na nossa modelagem a coluna `DataCadastro` e o tipo dela é `Date` .

```
SQL> alter table Produto add(DataCadastro Date);
```

Ao executarmos o comando, a tabela será alterada.

```
SQL> select table_name from User_tables;

TABLE_NAME
--------------------------------------------------------------------------------
PRODUTO

SQL> alter table Produto add(DataCadastro Date);

Tabela alterada.

SQL>
```

Para vermos as alterações, usaremos o comando `desc Produto` :

```
SQL> desc Produto;

 Nome                                                  Nulo?    Tipo
 ----------------------------------------------------- -------- ------------------------------
 ID                                                    NOT NULL NUMBER(10)
 NOME                                                           VARCHAR2(250)
 PRECO                                                          NUMBER(10,2)
 DATACADASTRO                                                   DATE

SQL>
```

Aprendemos a transformar uma entidade, que é a representação dos dados no modelo lógico, em um modelo físico, a tabela. Também vimos como alterá-la.

Iremos aprender a trabalhar com a tabela, que é o foco do nosso curso: como aproveitar os dados, como ordená-los e filtrá-los. Em seguida, iremos nos aprofundar sobre o comando `SELECT` .

Select

Aprendemos como criar uma tabela, mas queremos saber como aproveitar os dados, para agilizar o aprendizado, irei disponibilizar um arquivo, com alguns dados para trabalharmos ao longo do curso. Para navegarmos até o desktop, usaremos o comando `cd` , e depois um `dir` para que seja listado tudo que temos:

```
C:\Users\Alura\Desktop>dir
 Volume in drive C is BOOTCAMP
 Volume Serial Number Is CE60-34C1

 Directory of C:\Users\Alura\Desktop

13/04/2016  03:26   <DIR>          .
13/04/2016  03:26   <DIR>          ..
10/04/2016  08:16   <DIR>          oracle
13/04/2016  03:26   <DIR>     1.044 sql-oracle-2.sql
               1 File(s)         1.044 bytes
               3 Dir(s) 359.946.399.744 bytes free

C:\Users\Alura\Desktop>
```

Veremos que temos o arquivo `sql-oracle-2.sql` , que contêm uma tabela com os dados que iremos trabalhar.

Então, vou acessar o banco de dados Oracle, através da pasta `Desktop` . Vamos iniciar o `sqlplus` .

```
c:\Users\Alura\Desktop>sqlplus
```

Depois iremos ligar com o usuário `alura` e informar a senha. Ele irá mostrar que estamos conectados. Quero executar o `sql-oracle-2.sql` . Para executarmos um arquivo de fora, usaremos o comando `start` e o nome do arquivo.

```
SQL> start sql-oracle-2.sql
```

O SQL *\Plus irá mostrar que criamos um tabela com várias linhas. E como saber qual tabela ele criou? Nós executamos anteriormente, o `select table_name from user_tables` . Com este comando, ele irá imprimir a tabela `PRODUTO` e `FUNCIONARIOS` .

```
SQL> select table_name from user_tables;

TABLE_NAME
------------------------------------------------------------------------------
PRODUTO
FUNCIONARIOS
```

Ele acabou de cria a tabela `FUNCIONARIOS` . Como não iremos mais trabalhar com a tabela `Funcionarios` , nós vamos deletá-la com o comando `drop table` , porque queremos dropar a tabela. Após a tabela ser eliminada, se eu executar novamente o meu `select table_name form user_tables` só veremos a tabela `Funcionarios` .

```
SQL> drop table produto;

Tabela eliminada.

SQL> select table_name from user_tables;

TABLE_NAME
```

```
------------------------------------------------------------------------
FUNCIONARIOS
```

Para conhecermos a estrutura da tabela funcionários, nós podemos usar o comando `desc` e o nome da tabela.

```
SQL> desc funcionario;

 Nome                                                  Nulo?    Tipo
 ----------------------------------------------------- -------- ------------------------------
 ID                                                    NOT NULL NUMBER(10)
 NOME                                                  NOT NULL VARCHAR2(255)
 SALARIO                                               NOT NULL NUMBER(10,2)
 VALE REFEICAO                                         NOT NULL DATE
 SQL>
```

Ele mostrou que na tabela funcionários, encontramos: `Id` , `Nome` , `Salario` e Vale_refeicao . Todos os dados serão numéricos, com exceção do nome. Agora, o que devemos fazer para começarmos a visualizar os dados? Para consultar os dados, usaremos o comando select . Nós o utilizamos para listar o nome das tabelas. Mas para listar os dados da tabela funcionarios , usaremos o select from funcionários . Antes de executá-lo, precisaremos especificar os dados de quais colunas queremos ter acesso. Para exibir todas os dados podemos adicionar *`(asterísco):

```
SQL> select * from funcionarios;
```

Ele irá informar os dados contidos em todas as colunas. Observe que ainda que o nome da tabela seja escrito com letras maiúscula ("FUNCIONARIOS"), e no `select` , eu tenha escrito em minúsculo ("funcionarios"). Dentro do Oracle, nomes de colunas e tabelas são ***case sensitive***, tanto em caixa alta ou baixa, ele irá considerar da mesma forma.

Então, o Oracle nos mostra os dados de diversos funcionários como o Kauan e o José:

```
            ID
--------------
NOME
--------------------------------------------------------------------------------
   SALARIO VALE_REFEICAO
---------- -------------
         1
Kauan
      1100           300

         2
José
      1200           300
        ID
----------
```

Porém, pode ser que queremos ter acesso a um dado específico, como o salário. Por isso, eu vou escrever um `select` específico.

```
SQL> select salario from funcionario
```

Neste caso, escrevemos propositalmente o nome da tabela de forma errônea. Se esquecemos uma letra do nome, quando executarmos o comando, o Oracle irá informar que `a tabela ou view não existe`. Mas, quando alteramos e escrevemos o nome corretamente, ele irá disponibilizar todos os salários.

```
SQL> select salario from funcionarios;
   SALARIO
----------
      1100
      1200
      1200
      3700
      6700
      1700
      1700
      3400
      6700
      1500

10 linhas selecionadas

SQL>
```

No entanto, ele não irá apresentar de quem são os respectivos salários. Para isto, precisaremos que ele mostre também o nome dos funcionários. Podemos fazer isto, usando o `select nome, salario from funcionarios`. Basta separar os campos desejados com uma `,` (vírgula) e ele irá imprimir o `nome` e o `salario`. Veremos dos dois primeiros:

```
NOME
--------------------------------------------------------------------------------
    SALARIO
-----------
Kauan
      1100

Jose
      1200
----------
```

Quando observamos a tabela, ficamos com dúvida se o salário referido é bruto, se já sofreu descontos, ou inclui o vale-refeição. O recomendável é que alteremos o nome da coluna para exibição. Por exemplo, podemos renomear a coluna para salário bruto, com outro `select`.

```
SQL> select salario as salarioBruto from funcionarios;
```

Neste caso, vale lembrar que alguns bancos de dados não devemos usar `''` (aspas) junto com o nome da coluna ('salarioBruto'). Ao executarmos o comando escrito corretamente, ele mostrara a coluna com o nome alterado.

```
SQL> select salario as salarioBruto from funcionarios;

SALARIOBRUTO
```

```
------------
      1100
      1200
      1200
      3700
      6700
      1700
      1700
      3400
      6700
      1500

10 linhas selecionadas

SQL>
```

Podemos eliminar a palavra `as` do comando, que ele ainda será interpretado corretamente.

```
SQL> select salario salarioBruto from funcionarios;
```

Como não adicionamos `,` entre `salario` e `salarioBruto`, ele entende que não se trata de duas colunas.

Em seguida, iremos pedir que ele exiba as colunas `salario` e `vale_refeicao`.

```
SQL> select salario,vale_refeicao from funcionarios;

  SALARIO  VALE_REFEICAO
---------- -------------
      1100           300
      1200           300
      1200           300
      3700           300
      6700             0
      1700           300
      1700           300
      3400           300
      6700             0
      1500           300

10 linhas selecionadas.

SQL
```

Temos a opção de saber o somatório das duas colunas. Podemos abrir a calculadora, e fazer os cálculos manualmente. Mas seria uma forma muito trabalhosa. Com o `select` também podemos fazer contas entre os campos numéricos.

```
SQL> select salario + salarioBruto from funcionarios;

SALARIO+VALE_REFEICAO
---------------------
                 1400
                 1500
                 1500
```

```
                 4000
                 6700
                 2000
                 2000
                 3700
                 6700
                 1800

10 linhas selecionadas.

SQL>
```

Mas eu não gostei do nome da coluna `SALARIO+VALE_REFEICAO` e irei alterá-lo para o "apelido" `CUSTO` .

```
SQL> select salario + vale_refeicao as custo from funcionarios;

     CUSTO
----------
      1400
      1500
      1500
      4000
      6700
      2000
      2000
      3700
      6700
      1800

10 linhas selecionadas.

SQL>
```

Podemos ainda calcular o custo anual, com o seguinte comando:

```
SQL> select salario + vale_refeicao custo, salario + vale_refeicao * 12 from funcionarios;
```

Tanto o salário, como o vale-refeição serão multiplicados por `12` , referente aos doze meses do ano. O gasto anual iremos chamar de `custoAnual` .

```
SQL> select salario + vale_refeicao custo, salario +vale_refeicao * 12 as custoAnual from funcionari

     CUSTO CUSTOANUAL
----------------------
      1400       4700
      1500       4800
      1500       4800
      4000       7300
      6700       6700
      2000       5300
      2000       5300
      3700       7000
      6700       6700
```

```
      1800        5100
```

Observe que temos um problema no nosso relatório. O valor do `custoAnual` ficou muito abaixo do que deveria ser! Isto aconteceu porque a ordem em que inclui os elementos afeta. Porque todos os operadores aritméticos (`+,-,*,/`) tem uma ordem definida no Oracle. A ordem é:

1) Multiplicação

2) Divisão

3) Soma e Subtração

4) Concatenação

Se queremos trocar a ordem das operações, fazemos da mesma forma na matemática, usaremos os `parênteses` .

```
SQL> select salario + vale_refeicao custo, (salario +vale_refeicao) * 12 as custoAnual from funciona
```

Ao executarmos, teremos o valores corretos do `custaAnual` .

```
 SQL> select salario + vale_refeicao custo, (salario +vale_refeicao) * 12 as custoAnual from funcior

     CUSTO CUSTOANUAL
---------- ----------
      1400      16800
      1500      18000
      1500      18000
      4000      48000
      6700      80400
      2000      24000
      2000      24000
      3700      44400
      6700      80400
      1800      21600

10 linhas selecionadas.
```

É preciso ficar atento com a posição dos parêntese para que o resultado não seja prejudicado. Também não podemos incluir um `as` dentro dos parênteses, porque apenas expressões matemáticas podem ficar dentro do parênteses. Se tentássemos também, adicionar um parênteses dentro do parênteses, o Oracle não aceitaria, e iria reclamar que não encontrou parêntese. Uma situação assim, por exemplo, não seria aceita:

```
SQL> select salario + (vale_refeicao custo, (salario +vale_refeicao *12)) as custoAnual from funcior

ERRO na linha 1:
ORA-00907: parente direito nao encontrado
```

Podemos querer calcular o custo diário. Para isto, iremos considerar que o mês tem 30 dias. Vamos dividir o `salario + vale_refeicao` por 30, usaremos o operador `/` .

```
SQL> select salario + vale_refeicao custo, (salario +vale_refeicao) / 30 as custoDiario from funcio

     CUSTO CUSTODIARIO
---------- ----------
      1400 46,6666667
      1500         50
      1500         50
      4000 133,333333
      6700 233,333333
      2000 66,6666667
      2000 66,6666667
      3700 123,333333
      6700 233,333333
      1800         60

10 linhas selecionadas.
```

Existe um outro operador que é executado antes de **todos**: o unário. Com ele nós iremos trocar o sinal de um coluna. Se quisermos calcular, menos o salário somado ao vale-refeição, e em seguida, dividiremos por 30, adicionaremos um sinal de menos na expressão dentro do parênteses.

```
SQL> select salario + vale_refeicao custo, (-salario +vale_refeicao) / 30 as custoDiario from funci

     CUSTO CUSTODIARIO
---------- ----------
      1400 -26,666667
      1500        -30
      1500        -30
      4000 -113,33333
      6700 -223,33333
      2000 -46,666667
      2000 -46,666667
      3700 -103,33333
      6700 -223,33333
      1800        -40

10 linhas selecionadas.
```

O valor do custo de cada dia diminuiu. Observe que ao trocarmos o sinal do `salario` , ele passou a ser o primeiro elemento executado - tento prioridade, em relação à multiplicação e divisão.

Então, nós já vimos a ordem de execução dos nossos operadores.

Agora, queremos que exebir o texto no front da aplicação, a palavra `Custo` , para o valor resultante de `salario +vale_refeiçao` . Nós iremos somar o texto com algum outro campo, usaremos o operador de **concatenação**. Para isto usaremos o duplo pipe `||` .

```
SQL> select 'Custo' || salario + vale_refeicao custo, (salario +vale_refeicao) / 30 as custoAnual f
```

Se executarmos a linha como está agora, ele irá retornar uma mensagem de erro, porque estamos tentando concatenar o texto com uma expressão inteira. Precisaremos adicionar novos parênteses.

```
SQL> select 'Custo: ' || (salario + vale_refeicao) custo, (salario +vale_refeicao) / 30 as custoAnua

CUSTO                                                      CUSTODIARIO
---------------------------------------------------------- -----------
Custo: 1400                                                46,66666667
Custo: 1500                                                        50
Custo: 1500                                                        50
Custo: 4000                                                133,3333333
Custo: 6700                                                223,3333333
Custo: 2000                                                66,66666667
Custo: 2000                                                66,66666667
Custo: 3700                                                123,3333333
Custo: 6700                                                223,3333333
Custo: 1800                                                        60

10 linhas selecionadas.

SQL>
```

Observe que para somarmos um texto com uma impressão, nós precisamos usar o operador de concatenação || (duplo pipe).

Desta forma, já conhecemos alguns detalhes do `select` . No próximo vídeo, iremos falar mais a respeito e melhorar os *selects* que escrevemos até agora.

Variações do comando SELECT

Iremos conhecer algumas variações sobre o SELECT, um comando muito importante do SQL.

Agora, eu quero selecionar o valor do vale-refeição de todos os funcionários. Para isto vou usar um SELECT:

```
SQL> select vale_refeicao from funcionarios;

VALE_REFEICAO
-------------
          300
          300
          300
          300
            0
          300
          300
          300
            0
          300

10 linhas selecionadas.

SQL>
```

Mas temos a possibilidade também de mostrar apenas uma ocorrência do valor de `vale_refeicao` , sem precisar repetir tantas vezes.

Para imprimir apenas uma ocorrência de resultados distintos, vou acrescentar ao comando a palavra-chave `distinct` .

```
SQL> select distinct vale_refeicao from funcionarios;

VALE_REFEICAO
-------------
          300
            0

SQL>
```

E se quisermos fazer o mesmo com duas colunas simultaneamente? Por exemplo, se quisermos que ele retorne apenas uma ocorrência distinta dos valores dos salários e vale-refeição, podemos tentar acrescentando `distinct` após o `select` e depois da primeira coluna.

```
SQL> select distinct salario, vale_refeicao from funcionarios;

   SALARIO VALE_REFEICAO
---------- -------------
      1100           300
      3700           300
      3400           300
      6700             0
      1200           300
      1700           300
      1500           300

7 linhas selecionadas.

SQL>
```

Porém, o `vale_refeicao` está vindo repetido. Será que se usarmos o `distinct` colocarmos as duas colunas entre parêntes, resolveremos o problema?

```
SQL> select distinct(salario, vale_refeicao) from funcionarios;
select distinct(salario, vale_refeicao) from funcionarios

ERRO na linha 1:
ORA-00907: parentese direito nao encontrado

SQL>
```

O DISTINCT não é uma função, nós não podemos passar parâmetros em parênteses, e só podemos usá-lo uma vez a cada *query*. Ele considera a linha inteira, caso ela seja distinta de outra, ele não irá retornar repetições. Ou seja, o `distinct` será aplicado à linha e não apenas à coluna, e por isso, o valor `300` da segunda coluna apareceu várias vezes. Se precisamos usá-lo em mais de uma coluna, será através de uma função ou concatenando os valores distintos. Veremos isto mais adiante.

Agora, voltaremos a seguinte situação: vamos calcular o `salario + vale_refeicao` e diremos que é o `custo`.

```
SQL> select salario + vale_refeicao as custo from funcionarios;

     CUSTO
----------
      1400
      1500
      1500
      4000
      6700
      2000
      2000
      3700
      6700
      1800

10 linhas selecionadas.

SQL>
```

Calcularemos também o salário anual, multiplicando o resultado de `salario + vale_refeicao` multiplicado por 12.

```
SQL> select salario + vale_refeicao as custo,(salario + vale_refeicao) * 12 from funcionarios;

 CUSTO     (SALARIO+VALE_REFEICAO) * 12
---------- ----------------------------
      1400                        16800
      1500                        18000
      1500                        18000
      4000                        48000
      6700                        80400
      2000                        24000
      2000                        24000
      3700                        44400
      6700                        80400
      1800                        21600

10 linhas selecionadas.
```

Se `salario+vale_refeicao` é igual a `custo`, porque não podemos simplesmente multiplicar `custo * 12`?

```
SQL> select salario + vale_refeicao as custo, custo * 12 from funcionarios;
```

Se tentarmos executar esta linha, ele retornará uma mensagem de erro, dizendo que o identificador é inválido. Isto acontece, porque quando aplicamos um apelido na coluna, ele só será aplicado após os resultados serem apresentados. Quando usamos o `custo * 12`, estamos calculando o resultado ao mesmo tempo em que ele está buscando por cada registro. Então, no momento em que ele calcula a segunda coluna do `select`, o apelido `custo` ainda não foi aplicado. Dentro das colunas do SELECT, não podemos aplicar o apelido, também conhecido como **alias**, para realizar outra operação. Precisaremos repetir `salario+vale_refeicao`.

Até aqui conseguimos selecionar resultados distintos e aprendemos que o **alias** não pode ser usado em outras colunas do SELECT. Mas, vamos imaginar a situação em que o nosso chefe precise de alguns dados dos funcionários e queira saber quem recebe uma salário maior do que R$ 3000. Temos a opção de executar um `select` que irá nos mostrar o salário de todos os funcionários, e podemos analisar cada um e identificar quais são superiores a R$3000. No entanto, se a empresa tiver 2 mil funcionários, esta não seria uma opção interessante. O melhor seria filtrar os resultados, adicionando uma nova cláusula do SELECT.

```
SQL> select salario from funcionarios where salario > 3000;
```

Especificamos assim, que queremos todos os funcionários **onde** (`where`) uma condição é aceita, no caso, o salário é `> 3000` .

```
SQL> select salario from funcionarios where salario > 3000;

   SALARIO
----------
      3700
      6700
      3400
      6700

SQL>
```

O SQL *\Plus trouxe apenas os salários maiores ao valor estipulado. Se quisermos salários menores a R$3000, basta usarmos o sinal de menor `<` e ele irá apresentar os resultados que atendem a condição.

```
SQL> select salario from funcionarios where salario < 3000;

   SALARIO
----------
      1100
      1200
      1200
      1700
      1700
      1500

6 linhas selecionadas.

SQL>
```

Se quisermos saber quais salários são diferentes de R$3000, poderíamos usar os sinais `<>` .

```
SQL> select salario from funcionarios where salario <> 3000;

   SALARIO
----------
      1100
      1200
      1200
      3700
```

```
       6700
       1700
       1700
       3400
       6700
       1500

10 linhas selecionadas.

SQL>
```

Se quisermos saber se existem salários iguais a R$3000, usaremos o sinal de igual `=`.

```
SQL> select salario from funcionarios where salario = 3000;

nao ha linhas selecionadas
```

Ele irá retornar que não encontrou salários que atendencem a condição.

Conseguimos aplicar um filtro, usando diversas comparações. Podemos incluir ainda as condições de quem ganha um salário maior ou igual (`>=`). Poderíamos também pesquisar por quem ganha menor ou igual (`<=`) a R$3000. Para identificarmos os salários diferentes usariamos o `!=`, que tería o mesmo efeito de `<>`.

Nós já aprendemos a filtrar alguns resultados. Mas voltaremos ao pedido do meu chefe. Ele pediu que eu indicasse quem recebe mais de R$3000, e para isto, preciso incluir mais o valor do vale-refeição. Então, precisarei filtrar a partir do `salario+vale_refeicao`.

```
SQL> select salario + vale_refeicao from funcionario where salario > 3000

SALARIO+VALE_REFEICAO
---------------------
                4000
                6700
                3700
                6700

SQL>
```

Porém, na condição estamos considerando apenas `salario > 3000`. Teríamos que também considerar o salário somado ao vale-refeição. Gostaria de fazer outra alteração e renomear o nome da coluna para `custo`. Mudaremos primeiro o nome da coluna.

```
SQL> select salario + vale_refeicao as custo from funcionarios where custo > 3000;

   CUSTO
--------
    4000
    6700
    3700
    6700
```

Depois, no `where` quero definir que o `custo` deverá ser maior do que R$3000.

```
SQL> select salario + vale_refeicao from funcionario where custo > 3000;
select salario + vale_refeicao as custo from funcionarios where custo > 3000

ERRO na linha 1:
ORA-00904: "CUSTO": identificador invalido

SQL>
```

Ele me dará uma resposta semelhante a que vimos antes: CUSTO é um identificado inválido. No `where` acontece o mesmo quando tentamos usar um apelido (`alias`) no nome de coluna. Nós não podemos usar o `alias` dentro do `where` , porque este também será executado enquanto filtramos os resultados, ou seja, o alias ainda não foi criado.

Quando eu fiz o exame de certificação, duas questões de `pegadinhas` eram sobre o uso do alias em situações assim.

Fique atento: Dentro do `where` , não usamos o alias (um apelido).

Para alterarmos a condição, teremos que usar `salario + vale_refeicao` :

```
SQL> select salario + vale_refeicao as custo from funcionarios where salario _ vale_refeicao > 3000;

   CUSTO
--------
    4000
    6700
    3700
    6700
```

Agora, a filtragem dos resultados estará correta. O problema está em que se alterarmos a *query*, teremos que lembrar de fazer as mudanças nas duas partes. Tambéd, se fizemos uma operação mais complexa do que a de soma, e precisassemos usar uma fórmula gigante... A solução seria copiar um trecho enorme de código de uma parte e repeti-lo no outro. É mais trabalhoso, porém, como `where` não aceita alias, seria necessario.

Nós aprendemos a filtrar os resultados e quando podemos ou não utilizar o alias. Mais adiante, iremos nos aprofundar sobre o SELECT.

Outros Filtros

Vamos aprender agora a selecionar alguns filtros diferentes.

Queremos que seja listados os salários da tabela funcionários.

```
SQL> select salario from funcionarios;

   SALARIO
----------
      1100
      1200
      1200
```

```
      3700
      6700
      1700
      1700
      3400
      6700
      1500

10 linhas selecionadas.

SQL>
```

Queremos listar quem ganha exatamente quem recebe o salário R$3700 ou R$1700. Para descobrir quem recebe o salário de R$6700, podemos usar um `where` com `salario = 6700` .

```
SQL> select salario from funcionarios where salario = 6700;

  SALARIO
---------
     6700
     6700
```

Temos duas referências, quero agora que os nomes dos funcionários apareçam.

```
SQL> select nome,salario from funcionarios where salario = 6700;

NOME
--------------------------------------------------------------------------------
    SALARIO
-----------
Thais
      6700
Igor
      6700
```

Quero descobrir também quem ganha R$3700.

```
SQL> select nome,salario from funcionarios where salario = 3700;

NOME
--------------------------------------------------------------------------------
    SALARIO
-----------
Leonardo
      3700
```

Eu fiz duas *queries*, no entanto, quero que as duas comparações sejam combinadas em um só *query*. Faremos isto, utlizando o `or` (que seginfica "ou").

```
SQL> select nome,salario from funcionarios where salario = 6700 or salario = 3700;
```

```
NOME
--------------------------------------------------------------------------------
    SALARIO
-----------
Leonardo
      3700
Thais
      6700
Igor
      6700
```

O SQL *\Plus trouxe o Leonardo, a Thais e o Igor. Porém, já estamos pesquisando por salários específicos, e seu quisermos ocultar os valores da exibição. Será que irá funcionar se usarmos como condição no `where` uma coluna que não será exibida? Sim, podemos.

```
SQL> select nome from funcionarios where salario = 6700 or salario = 3700;

NOME
--------------------------------------------------------------------------------
Leonardo
Thais
Igor
```

Ele listou apenas o nome dos tres funcionários. Então, é possível combinar condições dentro do `where` , usando o `or` . Além disso, podemos usar como condição qualquer coluna, independente que ela seja exibida. Podemos pesquisar por um funcinário que recebe o salário de R$6700 e o vale-refeição de R$300. Para isto, iremos usar `and` .

```
SQL> select nome from funcionarios where salario = 6700 and vale_refeicao = 300;
```

Porém, neste caso, ele irá responder que não há linhas selecionadas. Mas, se nossa condição for alguém que receba R$6700 e não receba vale-refeição, o resultado será diferente.

```
SQL> select nome from funcionarios where salario = 6700 and vale_refeicao =0;

NOME
--------------------------------------------------------------------------------
Thais
Igor
```

Além de compararmos com `and` , na qual as duas condições precisam ser verdadeiras, também podemos comparar com `or` , e uma **ou** outra poderão ser verdadeiras. Vale lembrar que podemos usar o AND junto com outros operadores. Podemos criar um `where` , em que o salário seja menor que R$6700 e o funcionário não receba vale-refeição.

```
SQL> select nome from funcionarios where salario < 6700 and vale_refeicao = 0;

não ha linhas selecionadas
```

Usando `and` , `or` e `where` já conseguimos executar diversas *queries*. Mas existem certas condições que se repetem bastante, por exemplo, saber se um funcionário recebe um valor ou outro. Para isto, temos algumas funções que podem nos ajudar.

Se queremos saber quem recebe o salário R$3700 ou R$6700, significa que o salário estará **dentro** destes valores, usaremos `in`.

```
SQL> select nome from funcionarios where salarios in(3700,6700);

NOME
--------------------------------------------------------------------------------
Leonardo
Thais
Igor
```

Observe que o resultado foi o mesmo de quando usamos o `or`.

Podemos usar algumas alternativas equivalente ao `in`. Uma delas é o `some`. Porém, teremos que usar juntamente o operador, neste caso utilizaremos `= some`.

```
SQL> select nome from funcionarios where salario = some(3700, 6700);

NOME
--------------------------------------------------------------------------------
Leonardo
Thais
Igor
```

Outra opção é o `any`, que usaremos da mesma forma que o `some`.

```
SQL> select nome from funcionarios where salario = any(3700,6700);

NOME
--------------------------------------------------------------------------------
Leonardo
Thais
Igor
```

Podemos ainda criar outro tipo de condições com `any`. Por exemplo, pesquisar quais salários são maiores do que os valores entre parêntes.

```
SQL> select nome from funcionarios where salario > any (3700, 6700);

NOME
--------------------------------------------------------------------------------
Thais
Igor
```

Existe o caso em que queremos descobrir quais salários não são iguais a R$3700 e R$6700. Neste caso, podemos usar o `not in`.

```
SQL> select nome from funcionarios where salario not in(3700,6700);
```

```
NOME
--------------------------------------------------------------------------------
Kauan
Jose
Eduarda
Nicole
Leticia
Joao
Diogo

7 linhas selecionadas.
```

Foram listados todos os funcionários que recebem salários diferentes dos valores específicados.

Usando estas funções, nós conseguimos escrever *queries* mais potentes e fazer comparações mais complexas.

Mais sobre filtragem

Vamos aprender mais sobre as condições que podemos ter dentro do `where`. Iremos listar novamente o nome de todos os funcionários.

```
SQL> select nome from funcionarios;

NOME
------------------------------------------------------------
Kaun
Jose
Eduarda
Leonardo
Thais
Nicole
Leticia
Joao
Igor
Diogo
Karlos

NOME
------------------------------------------------------------
Stevan
Otavio

13 linhas selecionadas.
```

Vamos imaginar que existisse um funcionário chamado Eduardo e outra chamada Eduarda. Se eu quisesse selecionar um dos nomes, eu poderia fazer com um SELECT:

```
SQL> select nome from funcionarios where nome = 'Eduarda';

NOME
------------------------------------------------------------
Eduarda
```

Podemos pesquisar os dois nomes ao mesmo tempo com `or`.

```
SQL> select nome from funcionarios where nome = 'Eduarda' or nome= 'Eduardo';

NOME
------------------------------------------------------------
Eduarda
```

Observe que a nossa *query* funciona, mas temos um operador mais adequado: `like`. Com este operador, o SQL *Plus mostrará todos os nomes parecidos com Eduarda.

```
SQL> select nome from funcionarios where nome like 'Eduarda';

NOME
------------------------------------------------------------
Eduarda
```

Se quiséssemos que ele retornasse tanto "Eduarda" ou "Eduardo", deveríamos escrever `like 'Eduard_'`. Desta forma, ele aceitará que a última letra após "Eduard" seja qualquer caracter.

```
SQL> select nome from funcionarios where nome like 'Eduard_';
```

Ele irá retornar novamente o nome da Eduarda. Mas, e se eu quisesse que fosse listado todos os funcionários que começassem com a letra "A"? Se tentassemos usar o `like A___________`, o resultado não seria o esperado. Porque o o número de `_` (*underlines*) precisaria ser exatamente o número de caracteres do nome. Para conseguir que ele liste todos os nomes que comecem com uma determinada letra devemos especificá-la seguida por `%`. Por exemplo, se testarmos `E%`, teremos os seguintes resultados:

```
SQL> select nome from funcionarios where nome like 'L%';

NOME
------------------------------------------------------------
Leonardo
Leticia
```

Se quisermos identificar todos os nomes que terminam com uma letra, basta inverter a posição do sinal de `%`.

```
SQL> select nome from funcionarios where nome like '%a';

NOME
------------------------------------------------------------
Eduarda
Leticia
```

O `like` nos permite fazer uma busca mais esperta. Outro tipo de pesquisa que podemos pensar é no caso em que queremos descobrir quais funcionários não recebem vale-refeição. Para isto, usaremos o `is null`.

```
SQL> select nome from funcionarios where vale_refeicao is null;

NOME
--------------------------------------------------------------------------------
Karlos
Stevan
Otavio
```

Nunca podemos comparar `null` usando o sinal de `=` ("igual"). Sempre iremos usar `is null` ou `is not null`.

Quando eu realizei o exame de certificação, várias questões foram sobre `null`. Quando formos criar o `where`, **não** podemos usar `<, >, =, !=`. Tenha muita atenção, porque este tema costuma cair como uma pegadinha no exame.

Observe que `null` é **diferente de zero**. Se pesquisassemos os funcionário que o `vale_refeicao is not null`, todos os funciocários seriam listados. Afinal, na tabela, quem recebe o vale-refeição tem o valor igual a 0.

Um último exemplo de uso do `like` que podemos usar é na situação de pesquisarmos um nome que tenha um caracter no meio como um *underline*(_), por exemplo, "lua*na". Para especificarmos que o* `` `seja considerado como texto, devemos acrescentar a instrução "ESCAPE" e o caracter "\". Iremos adicionar ` `` antes do *underline* no texto.

```
SQL> select nome from funcionarios where nome like 'Lua\_na" ESCAPE '\';
```

Ele irá pesquisar o termo `Lua_na`. Usaremos o `ESCAPE` sempre que quisermos tratar um caracter especial com normal.

Mais adiante, iremos pesquisar a ordem dos resultados, como ela vem do banco de dados. Até a próxima aula!