

# Oracle

Conteúdos relacionados ao banco Oracle (Configuração, SQL's de apoio, etc.)

- Criação de View
- Criar usuário somente para consultas no banco de dados
- Descobrir SQL's executados (V\$SQL)
- Função - Converter BLOB para CLOB
- Função - Converter CLOB para BLOB
- Função - Formata CNPJ
- Função - Formata CPF
- Função - Gera MD5
- Função - Remove acentos
- Função - Remove caractere especial
- Função - Retorna apenas números
- Função - Retorna lista
- Função - Retorna parte de um campo de texto/clob definindo início e fim
- Função - Retorna períodos semanais do mês (domingo à sábado)
- Função - Retorna quantidade de dias úteis
- Função - Retorna valor por extenso
- Funções de agregação (count, sum, max, min, avg, median, first, last, partition by)
- Nvl e Coalesce
- Operações Join
- Order by, Group by, Having e Distinct
- Pacote de Funções DOX (Package) - Oracle
- Retorna datas de um intervalo de dias
- Retorna tempo de um intervalo de datas
- SQL - Descobrir se existem registros duplicados
- SQL - Usuários com licenças do DOX Portal
- Union, Union All, Minus e Intersect
- Utilização de Datas e Conversões (current\_date, current\_time, timestamp, etc)

- With, Substr e Instr

# Criação de View

A view é útil para ter uma consulta pré-organizada que é executada frequentemente.

## Exemplo

Neste exemplo, temos uma tabela chamada CIDADE com informações de várias cidades de alguns estados, incluindo um campo que informa se o registro está inativo.

IDCIDADE	DESCRICAO	UF	INATIVO
0	Desconhecido	SC	S
1	Criciúma	SC	N
2	São Paulo	SP	N
3	Balneário Rincão	SC	N
4	Porto Alegre	RS	N
5	Araranguá	SC	N
6	Içara	SC	N
7	Curitiba	PR	N
8	Torres	RS	N
9	Florianópolis	SC	N

Precisamos que na consulta, seja mostrado apenas cidades que não estão inativas com o código e a descrição concatenada com a UF (Ex: Florianópolis - SC). Além disso, só mostrar as cidades do estado SC e ordenado pela descrição.

```
CREATE OR REPLACE VIEW VW_CIDADE_SC AS
SELECT IDCIDADE AS CODIGO,
       DESCRICAO||' - '||UF AS DESCRICAO
FROM CIDADE
WHERE UF = 'SC'
      AND INATIVO = 'N'
ORDER BY DESCRICAO
```

## Chamada da view:

```
SELECT *
FROM VW_CIDADE_SC
```

Retorno:

CODIGO	DESCRICAO
5	Araranguá - SC
3	Balneário Rincão - SC
1	Criciúma - SC
9	Florianópolis - SC
6	Içara - SC

# Criar usuário somente para consultas no banco de dados

Quando for necessário criar um usuário no banco de dados de consulta das tabelas da Ema, executar os comandos abaixo:

## 1 - Criação do usuário:

```
CREATE USER DOX_CONSULTA
  IDENTIFIED BY PASSWORD
  DEFAULT TABLESPACE USERS
  TEMPORARY TABLESPACE TEMP
  PROFILE DEFAULT
  ACCOUNT UNLOCK;

declare
  va_sql varchar2(1000);
  va_scm varchar2(50);
  va_usr varchar2(50);
  va_typ varchar2(50);

begin
  va_scm := 'EMA';
  va_usr := 'DOX_CONSULTA';

  for reg in (select a.OBJECT_NAME, a.OBJECT_TYPE
              from ALL_OBJECTS a
              where a.OWNER = va_scm
              and a.OBJECT_TYPE in (select distinct(OBJECT_TYPE) from
ALL_OBJECTS)
              and a.OBJECT_TYPE <> ' TRIGGER'
              and a.OBJECT_TYPE <> ' INDEX'
              AND A.OBJECT_TYPE <> ' SEQUENCE'
              AND A.OBJECT_TYPE <> ' TYPE'
              AND A.OBJECT_TYPE <> ' LOB'
```

```

) loop
    begin
        if (reg.OBJECT_TYPE = 'FUNCTION' or reg.OBJECT_TYPE = 'PROCEDURE' or reg.OBJECT_TYPE =
'PACKAGE' or reg.OBJECT_TYPE = 'PACKAGE BODY') then
            va_typ := 'EXECUTE';
        else
            va_typ := 'SELECT';
        end if;
        va_sql := 'grant ' || va_typ || ' on ' || va_scm || '.' || reg.OBJECT_NAME || ' to ' ||
va_usr;
        execute immediate va_sql;

        exception
            when no_data_found then
                null;
            when others then
                dbms_output.put_line(sqlerrm);
                null;
        end;
    end loop;
end;
/

```

## 2 - Permissão:

```
grant create session to DOX_CONSULTA;
```

# Descobrir SQL's executados (V\$SQL)

**V\$SQL:** É uma tabela de consulta dos SQL executados sem GROUP BY, normalmente são atualizadas após a execução de uma consulta. Porém, para consultas de longa execução, elas são atualizadas a cada 5 minutos.

Isso facilita a visualização do impacto das instruções SQL de execução longa enquanto eles ainda estão em andamento. Pode ser utilizado, por exemplo, para descobrir qual SQL está sendo executado em um Kanban.

```
SELECT *  
FROM V$SQL
```

Exemplo:

[image-1646855117085.png](#)

Image not found or type unknown

No Kanban em que deseja descobrir os SQL sendo executados filtre algum dado que exista, neste caso usamos o cliente. Assim que pesquisar use o SQL abaixo para ver os últimos SQL executados.

Para filtrar mais a consulta retornada você pode adicionar condições ao SQL do v\$SQL como o like, segue o exemplo para esse caso.

```
SELECT *  
FROM V$SQL  
WHERE SQL_FULLTEXT LIKE '%288%'  
AND SQL_FULLTEXT LIKE '%count%'  
AND SQL_FULLTEXT LIKE '%12069%'
```

No primeiro LIKE está sendo filtrado o número do procedimento do kanban, no segundo é o campo em que deseja descobrir, neste caso queremos saber a quantidade dos card's na coluna e por fim o código do cliente executado no filtro.

image-1646855448874.png

Image not found or type unknown

Com dois clique no ultimo SQL é possível copiar o código e executa-lo.

image-1646855474930.png

Image not found or type unknown

Assim traz o resultado de registros, está função pode ser feita para qualquer outro SQL executa em banco ORACLE.

Homologado: **ORACLE - 12.5**



# Função - Converter BLOB para CLOB

## SQL:

```
create or replace function blob_to_clob (b in blob) return clob is
    pos      pls_integer := 1;
    buffer    varchar2( 32767 );
    res       clob;
    lob_len   pls_integer := dbms_lob.getlength( b );
begin
    dbms_lob.createtemporary( res, true );
    dbms_lob.open( res, dbms_lob.lob_readwrite );

    loop
        buffer := utl_raw.cast_to_varchar2( dbms_lob.substr( b, 16000, pos ) );

        if length( buffer ) > 0 then
            dbms_lob.writeappend( res, length( buffer ), buffer );
        end if;

        pos := pos + 16000;

        exit when pos > lob_len;
    end loop;

    return res;
end blob_to_clob;
```

## Chamada da função:

```
select blob_to_clob(utl_raw.cast_to_raw(' Valor Blob' )) from dual
```

## Retorno:

Valor Blob

# Função - Converter CLOB para BLOB

## SQL:

```
create or replace function clob_to_blob(l_clob clob) return blob is
    l_blob          blob;
    l_dest_offset    number := 1;
    l_src_offset     number := 1;
    l_lang_context    number := dbms_lob.default_lang_ctx;
    l_warning         number;
begin
    dbms_lob.createtemporary(l_blob, true);
    dbms_lob.converttoblob(dest_lob      => l_blob,
                           src_clob      => l_clob,
                           amount        => dbms_lob.lobmaxsize,
                           dest_offset   => l_dest_offset,
                           src_offset    => l_src_offset,
                           blob_csid     => nls_charset_id('al32utf8'),
                           lang_context  => l_lang_context,
                           warning       => l_warning);

    return l_blob;
end clob_to_blob;
```

## Chamada da função:

```
select clob_to_blob(' Teste' ) from dual
```

## Retorno:

```
( BLOB)
```

# Função - Formata CNPJ

## SQL:

```
create or replace function formata_cnpj (xnpj in varchar2) return varchar2 is
    retorno varchar2(18);
begin

    select substr(lpad(xnpj, 14, '0'),1,2) || '.' ||
           substr(lpad(xnpj, 14, '0'),3,3) || '.' ||
           substr(lpad(xnpj, 14, '0'),6,3) || '/' ||
           substr(lpad(xnpj, 14, '0'),9,4) || '-' ||
           substr(lpad(xnpj, 14, '0'),13,2) into retorno
    from dual;

    return retorno;
end;
```

## Chamada da função:

```
select formata_cnpj('12345678000910') from dual
```

## Retorno:

```
12. 345. 678/0009-10
```

# Função - Formata CPF

## SQL:

```
create or replace function formata_cpf (xcpf in varchar2) return varchar2 is
    retorno varchar2(14);
begin
    select substr(lpad(xcpf, 11, '0'),1,3) || '.' ||
           substr(lpad(xcpf, 11, '0'),4,3) || '.' ||
           substr(lpad(xcpf, 11, '0'),7,3) || '-' ||
           substr(lpad(xcpf, 11, '0'),10,2) into retorno
    from dual;

    return retorno;
end;
```

## Chamada da função:

```
select formata_cpf('12345678910') from dual
```

## Retorno:

```
123. 456. 789-10
```

# Função - Gera MD5

## SQL:

```
create or replace function md5 (valor varchar) return varchar2 is
  v_input  varchar2(2000) := valor;
  hexkey   varchar2(32) := null;
begin
  hexkey := rawtohex(dbms_obfuscation_toolkit.md5(input => utl_raw.cast_to_raw(v_input)));
  return nvl (hexkey, '');
end;
```

## Chamada da função:

```
select md5(' Teste' ) from dual
```

## Retorno:

```
8E6F6F815B50F474CF0DC22D4F400725
```

# Função - Remove acentos

## SQL:

```
create or replace function remove_acento (i_texto in varchar2) return varchar2
is v_texto      varchar2( 32767);
begin
  v_texto := i_texto;
  v_texto := replace(v_texto,' Ã ', ' A' );
  v_texto := replace(v_texto,' Õ ', ' O' );
  v_texto := replace(v_texto,' ã ', ' a' );
  v_texto := replace(v_texto,' õ ', ' o' );
  v_texto := replace(v_texto,' Á ', ' A' );
  v_texto := replace(v_texto,' É ', ' E' );
  v_texto := replace(v_texto,' Í ', ' I' );
  v_texto := replace(v_texto,' Ó ', ' O' );
  v_texto := replace(v_texto,' Ú ', ' U' );
  v_texto := replace(v_texto,' á ', ' a' );
  v_texto := replace(v_texto,' é ', ' e' );
  v_texto := replace(v_texto,' í ', ' i' );
  v_texto := replace(v_texto,' ó ', ' o' );
  v_texto := replace(v_texto,' ú ', ' u' );
  v_texto := replace(v_texto,' À ', ' A' );
  v_texto := replace(v_texto,' à ', ' a' );
  v_texto := replace(v_texto,' Â ', ' A' );
  v_texto := replace(v_texto,' Ê ', ' E' );
  v_texto := replace(v_texto,' Ô ', ' O' );
  v_texto := replace(v_texto,' â ', ' a' );
  v_texto := replace(v_texto,' ê ', ' e' );
  v_texto := replace(v_texto,' ô ', ' o' );
  v_texto := replace(v_texto,' ç ', ' c' );
  v_texto := replace(v_texto,' Ç ', ' C' );
  v_texto := replace(v_texto,' ü ', ' u' );
  v_texto := replace(v_texto,' Ü ', ' U' );
  v_texto := replace(v_texto,' ò ', ' .' );
  v_texto := replace(v_texto,' ã ', ' .' );
  v_texto := replace(v_texto,' ; ', ' ' );
```

```
v_texto := replace(v_texto,'&',' ');  
v_texto := replace(v_texto,chr(39),' ');  
return v_texto;  
end;
```

### **Chamada da função:**

```
select remove_acento(' Ã ê I ó Ü' ) from dual
```

### **Retorno:**

```
A e I o U
```



# Função - Remove caractere especial

## SQL:

```
create or replace function remove_char_esp(xvalor in varchar) return varchar2 deterministic is
begin
    return trim(replace(translate(xvalor,'_ = . [ ] ; > < - / ! @ # $ % ^ & * ( ) \ + { }', ' '), ' ', ''));
end;
```

## Chamada da função:

```
select remove_char_esp('Caracter (-@-) Especial!') from dual
```

## Retorno:

```
CaracterEspecial
```

# Função - Retorna apenas números

## SQL:

```
create or replace function retorna_apenas_numero(xvalor in varchar) return varchar2
deterministic is
begin
    return trim(replace(translate(xvalor, ' -
    /!@#%` &* <>{ } [ ] = + ( ) \ +abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZÁÇÉÍÓÚáâêôãöüÁÇÉÍÓÚÂÊ
    ', ' '), ' ', ''));
end;
```

## Chamada da função:

```
select retorna_apenas_numero('1 a 2 b 3 c') from dual
```

## Retorno:

```
123
```

# Função - Retorna lista

## SQL's:

```
create or replace package pkg_retorna_linhas as
  type linhas is table of varchar2(4000);
  function retorna_linhas(texto clob, delimitador varchar2) return linhas pipelined;
end pkg_retorna_linhas;
```

```
create or replace package body pkg_retorna_linhas as
  function retorna_linhas (texto clob, delimitador varchar2) return linhas pipelined as
    linha clob;
    contador integer := 1;
    limite integer;
  begin
    loop
      limite := instr(substr(texto, contador), delimitador);
      if (limite = 0 and contador <= length(texto)) then
        limite := length(texto);
      end if;
      linha := trim(substr(substr(texto, contador), 1, limite));
      contador := contador+limite;
      exit when linha is null;
      pipe row(replace(linha, delimitador, ''));
    end loop;
  end retorna_linhas;
end pkg_retorna_linhas;
```

## Chamada da função:

```
select column_value as lista
  from table(pkg_retorna_linhas.retorna_linhas('1,2,3', ','))
```

## Retorno:

```
| LISTA |
-----
```

	1	
	2	
	3	

# Função - Retorna parte de um campo de texto/clob definindo inicio e fim

## SQL:

```
create or replace function retorna_parte (xtexto in clob, xdesde in varchar2, xate in
varchar2, xcont in integer) return varchar2 is
    info varchar2(2000);
begin
    select trim(substr(xtexto, instr(xtexto, xdesde, 1, xcont)+length(xdesde), instr(xtexto,
xate, instr(xtexto, xdesde, 1, xcont)+length(xdesde)) - (instr(xtexto, xdesde, 1,
xcont)+length(xdesde))))
        into info from dual;
    return info;
end;
```

## Chamada da função:

```
select retorna_parte(' <nome>João da
Silva</nome><dtnascimento>30/03/1990</dtnascimento><profissao>Consultor D0X</profissao>',
' <nome>', '</', 1) as nome,
    retorna_parte(' <nome>João da
Silva</nome><dtnascimento>30/03/1990</dtnascimento><profissao>Consultor D0X</profissao>',
' <dtnascimento>', '</', 1) as dtnascimento,
    retorna_parte(' <nome>João da
Silva</nome><dtnascimento>30/03/1990</dtnascimento><profissao>Consultor D0X</profissao>',
' <profissao>', '</', 1) as profissao
from dual
```

## Retorno:

NOME	DATA	PROFISSAO
-----		



# Função - Retorna períodos semanais do mês (domingo à sabado)

Essa função pode ser usada para casos em que é necessário coletar dados de uma base, onde será somado um valor agrupado por semana. Neste caso, o retorno da função será usado como filtro na consulta.

Exemplo:

<b>Semana 1: R\$ 100</b>
<b>Semana 2: R\$: 188</b>

**Primeiramente criar a TYPE para que a função retorne uma tabela com várias colunas:**

```
CREATE TYPE t_semanas_linha AS OBJECT (id integer, datainicio date, datafim date);  
CREATE TYPE t_semanas_tabela IS TABLE OF t_semanas_linha;
```

**Em seguida criar a função:**

```
CREATE OR REPLACE FUNCTION f_semanas_mes (mesano varchar2) RETURN t_semanas_tabela AS  
    linha_tabela  
    t_semanas_tabela := t_semanas_tabela();  
    datainicio date;  
    datafim date;  
    contador integer := 1;  
BEGIN  
    FOR i IN  
        (SELECT dataatual, TO_CHAR(dataatual, 'D') as diasemana, LAST_DAY(dataatual)  
        ultimodia  
        FROM (select data_inicial + level - 1 dataatual  
        from (select to_date('01/' || mesano, 'DD/MM/YYYY') data_inicial from  
        dual)
```

```

        connect by level <= last_day(to_date(mesano, 'MM/YYYY')) - data_inicial + 1))
LOOP
    if extract(day from i.dataatual) = 1 or i.diasemana = 1 then
        datainicio := i.dataatual;
    end if;

    if i.dataatual = i.ultimodia or i.diasemana = 7 then
        datafim := i.dataatual;
        linha_tabela.extend;
        linha_tabela(linha_tabela.last) := t_semanas_linha(contador, datainicio,
datafim);
        contador := contador + 1;
    end if;

END LOOP;

RETURN linha_tabela;
END;
```

### Chamada da função:

```

SELECT *
FROM TABLE( f_semanas_mes(' 02/2020' ) )
```

### Retorno:

ID	DATAINICIO	DATAFIM
1	01/02/2020	01/02/2020
2	02/02/2020	08/02/2020
3	09/02/2020	15/02/2020
4	16/02/2020	22/02/2020
5	23/02/2020	29/02/2020



# Função - Retorna quantidade de dias úteis

## SQL:

```
create or replace function dias_uteis (vdatai in date, vdataf in date) return varchar2 as
    total_dias number;
    total_feritados number;
    total_dias_uteis number := 0;
begin
    if vdatai is not null then
        /* conta a quantidade de dias entre as datas sem verificar feriados, sábados e domingos
        */
        for cur_row in (select to_date(vdataf)-to_date(vdatai) ttdias from dual) loop
            total_dias:=cur_row.ttdias;
        end loop;

        /* percorre dia por dia sempre checando se deseja contar ou não sabado e domingo*/
        for i in 0 .. total_dias loop
            for cur_row in (select (case when to_char(to_date(vdatai)+i,'D') not in(1,7) then 1
            else 0 end) tt from dual) loop
                total_dias_uteis:=total_dias_uteis+cur_row.tt;
            end loop;
        end loop;

        total_dias:=total_dias+1;
    else
        total_dias_uteis := null;
    end if;

    return total_dias_uteis;
end;
```

## Chamada da função:

---

```
select dias_uteis(to_date('01/07/2019', 'DD/MM/YYYY'), sysdate) from dual
```

**Retorno:**

13

# Função - Retorna valor por extenso

## SQL:

```
create or replace function valor_extenso (valor number) return varchar2 is
    extenso varchar2(240); b1 number(1); b2 number(1); b3 number(1); b4 number(1); b5
number(1); b6 number(1);
    b7 number(1); b8 number(1); b9 number(1); b10 number(1); b11 number(1); b12 number(1); b13
number(1); b14 number(1);
    l1 varchar2(12); l2 varchar2(3); l3 varchar2(9); l4 varchar2(3); l5 varchar2(6); l6
varchar2(8); l7 varchar2(12);
    l8 varchar2(3); l9 varchar2(9); l10 varchar2(3); l11 varchar2(6); l12 varchar2(8); l13
varchar2(12); l14 varchar2(3);
    l15 varchar2(9); l16 varchar2(3); l17 varchar2(6); l18 varchar2(8); l19 varchar2(12); l20
varchar2(3); l21 varchar2(9);
    l22 varchar2(3); l23 varchar2(6); l24 varchar2(16); l25 varchar2(3); l26 varchar2(9); l27
varchar2(3); l28 varchar2(6);
    l29 varchar2(17); virgula_bi char(3); virgula_mi char(3); virgula_mil char(3); virgula_cr
char(3); valor1 char(14);
-- TABELA DE CENTENAS --
    centenas char(108) := '          Cento    Duzentos   Trezentos' ||
                          ' Quatrocentos  Quinhentos  Seiscentos' ||
                          '  Setecentos  Oitocentos  Novecentos';
-- TABELA DE DEZENAS --
    dezenas char(79)  := '          Dez     Vinte    Trinta  Quarenta' ||
                          ' Cinquenta  Sessenta   Setenta  Oitenta' ||
                          ' Noventa';
-- TABELA DE UNIDADES --
    unidades char(54) := '          Um  Dois   TresQuatro  Cinco   Seis' ||
                          '   Sete  Oito   Nove';
-- TABELA DE UNIDADES DA DEZENA 10 --
    unid10 char(81) := '          Onze     Doze     Treze  Quatorze' ||
                       '   QuinzeDezesseisDezessete  Dezoito' ||
                       ' Dezenove';
```

```

begin
    valor1 := lpad(to_char(valor*100), 14, '0');
    b1 := substr(valor1, 1, 1);
    b2 := substr(valor1, 2, 1);
    b3 := substr(valor1, 3, 1);
    b4 := substr(valor1, 4, 1);
    b5 := substr(valor1, 5, 1);
    b6 := substr(valor1, 6, 1);
    b7 := substr(valor1, 7, 1);
    b8 := substr(valor1, 8, 1);
    b9 := substr(valor1, 9, 1);
    b10 := substr(valor1, 10, 1);
    b11 := substr(valor1, 11, 1);
    b12 := substr(valor1, 12, 1);
    b13 := substr(valor1, 13, 1);
    b14 := substr(valor1, 14, 1);
    if valor != 0 then
        if b1 != 0 then
            if b1 = 1 then
                if b2 = 0 and b3 = 0 then
                    l5 := 'Cem';
                else
                    l1 := substr(centenas, b1*12-11, 12);
                end if;
            else
                l1 := substr(centenas, b1*12-11, 12);
            end if;
        end if;
        if b2 != 0 then
            if b2 = 1 then
                if b3 = 0 then
                    l5 := 'Dez';
                else
                    l3 := substr(unid10, b3*9-8, 9);
                end if;
            else
                l3 := substr(dezenas, b2*9-8, 9);
            end if;
        end if;
        if b3 != 0 then

```

```

        if b2 != 1 then
            l5 := substr(unidades, b3*6-5, 6);
        end if;
    end if;
if b1 != 0 or b2 != 0 or b3 != 0 then
    if (b1 = 0 and b2 = 0) and b3 = 1 then
        l5 := 'Um';
        l6 := 'Bilhão';
    else
        l6 := 'Bilhões';
    end if;
if valor > 999999999 then
    virgula_bi := ' e ';
    if (b4+b5+b6+b7+b8+b9+b10+b11+b12) = 0 then
        virgula_bi := ' de' ;
    end if;
end if;
l1 := ltrim(l1);
l3 := ltrim(l3);
l5 := ltrim(l5);
if b2 > 1 and b3 > 0 then
    l4 := ' e ';
end if;
if b1 != 0 and (b2 != 0 or b3 != 0) then
    l2 := ' e ';
end if;
end if;
-- ROTINA DOS MILHOES --
if b4 != 0 then
    if b4 = 1 then
        if b5 = 0 and b6 = 0 then
            l7 := 'Cem';
        else
            l7 := substr(centenas, b4*12-11, 12);
        end if;
    else
        l7 := substr(centenas, b4*12-11, 12);
    end if;
end if;
if b5 != 0 then

```

```

if b5 = 1 then
    if b6 = 0 then
        l11 := 'Dez';
    else
        l9 := substr(unid10, b6*9-8, 9);
    end if;
else
    l9 := substr(dezenas, b5*9-8, 9);
end if;
end if;
if b6 != 0 then
    if b5 != 1 then
        l11 := substr(unidades, b6*6-5, 6);
    end if;
end if;
if b4 != 0 or b5 != 0 or b6 != 0 then
    if (b4 = 0 and b5 = 0) and b6 = 1 then
        l11 := ' Um';
        l12 := ' Milhão';
    else
        l12 := ' Milhões';
    end if;
if valor > 999999 then
    virgula_mi := ' e ';
    if (b7+b8+b9+b10+b11+b12) = 0 then
        virgula_mi := ' de ';
    end if;
end if;
l7 := ltrim(l7);
l9 := ltrim(l9);
l11 := ltrim(l11);
if b5 > 1 and b6 > 0 then
    l10 := ' e ';
end if;
if b4 != 0 and (b5 != 0 or b6 != 0) then
    l8 := ' e ';
end if;
end if;
-- ROTINA DOS MILHARES --
if b7 != 0 then

```

```

if b7 = 1 then
    if b8 = 0 and b9 = 0 then
        l17 := 'Cem';
    else
        l13 := substr(centenas, b7*12-11, 12);
    end if;
else
    l13 := substr(centenas, b7*12-11, 12);
end if;

end if;

if b8 != 0 then
    if b8 = 1 then
        if b9 = 0 then
            l17 := 'Dez';
        else
            l15 := substr(unid10, b9*9-8, 9);
        end if;
    else
        l15 := substr(dezenas, b8*9-8, 9);
    end if;
end if;

if b9 != 0 then
    if b8 != 1 then
        l17 := substr(unidades, b9*6-5, 6);
    end if;
end if;

if b7 != 0 or b8 != 0 or b9 != 0 then
    if (b7 = 0 and b8 = 0) and b9 = 1 then
        l17 := 'Um';
        l18 := ' Mil';
    else
        l18 := ' Mil';
    end if;

    if valor > 999 and (b10+b11+b12) != 0 then
        virgula_mil := ' e ';
    end if;

    l13 := ltrim(l13);
    l15 := ltrim(l15);
    l17 := ltrim(l17);

    if b8 > 1 and b9 > 0 then

```

```

        l16 := ' e ';
    end if;
    if b7 != 0 and (b8 != 0 or b9 != 0) then
        l14 := ' e ';
    end if;
end if;
-- ROTINA DOS REAIS --
if b10 != 0 then
    if b10 = 1 then
        if b11 = 0 and b12 = 0 then
            l19 := ' Cem';
        else
            l19 := substr(centenas, b10*12-11, 12);
        end if;
    else
        l19 := substr(centenas, b10*12-11, 12);
    end if;
end if;
if b11 != 0 then
    if b11 = 1 then
        if b12 = 0 then
            l23 := ' Dez';
        else
            l21 := substr(unid10, b12*9-8, 9);
        end if;
    else
        l21 := substr(dezenas, b11*9-8, 9);
    end if;
end if;
if b12 != 0 then
    if b11 != 1 then
        l23 := substr(unidades, b12*6-5, 6);
    end if;
end if;
if b10 != 0 or b11 != 0 or b12 != 0 then
    if valor > 0 and valor < 2 then
        l23 := ' Um';
    end if;
    l19 := ltrim(l19);
    l21 := ltrim(l21);

```



```

l23 := ltrim(l23);
if b11 > 1 and b12 > 0 then
    l22 := ' e ';
end if;
if b10 != 0 and (b11 !=0 or b12 != 0) then
    l20 := ' e ';
end if;
end if;
if valor > 0 and valor < 2 then
    if b12!=0 then
        l24 := ' Real';
    end if;
else
    if valor > 1 then
        l24 := ' Reais';
    end if;
end if;
-- TRATA CENTAVOS --
if b13 != 0 OR b14 != 0 then
    if valor > 0 then
        if (b12 != 0) or (b1+b2+b3+b4+b5+b6+b7+b8+b9+b10+b11+b12)!=0 then
            L25 := ' e ';
        end if;
    end if;
if b13 != 0 then
    if b13 = 1 then
        if b14 = 0 then
            l28 := ' Dez';
        else
            l26 := substr(unid10, b14*9-8, 9);
        end if;
    else
        l26 := substr(dezenas, b13*9-8, 9);
    end if;
end if;
if b14 != 0 then
    if b13 != 1 then
        l28 := substr(unidades, b14*6-5, 6);
    end if;
end if;

```

```

if b13 != 0 or b14 != 0 then
    if valor = 1 then
        l28 := 'Um';
    end if;
    l26 := ltrim(l26);
    l28 := ltrim(l28);
    if b13 > 1 and b14 > 0 then
        l27 := ' e ';
    end if;
end if;
if (b1+b2+b3+b4+b5+b6+b7+b8+b9+b10+b11+b12) > 0
then
    if b13 = 0 and b14 = 1 then
        l29 := ' Centavo';
    else
        l29 := ' Centavos';
    end if;
else
    if b13 = 0 and b14 = 1 then
        l29 := ' Centavo de Real';
    else
        l29 := ' Centavos de Real';
    end if;
end if;
-- CONCATENAR O LITERAL --
if l29 = ' Centavo de Real' or l29 = ' Centavos de Real' then
    virgula_mil := '';
end if;

extenso := l1|l2|l3|l4|l5|l6|virgula_bi
        ||l7|l8|l9|l10|l11|l12|virgula_mi
        ||l13|l14|l15|l16|l17|l18|virgula_mil
        ||l19|l20|l21|l22|l23|l24|virgula_cr
        ||l25|l26|l27|l28|l29;

extenso := ltrim(extenso);
extenso := replace(extenso, ' ', ' ');
else
    extenso := ' Zero';

```

```
end if;  
return extenso;  
end;
```

### **Chamada da função:**

```
select valor_extenso(1452.45) from dual
```

### **Retorno:**

Um Mil e Quatrocentos e Cinquenta e Dois Reais e Quarenta e Cinco Centavos

# Funções de agregação (count, sum, max, min, avg, median, first, last, partition by)

Confira algumas funções que ajudam nas consultas, onde é necessário criar agrupamentos de diferentes formas.

## TABELA EXEMPLO

### EX\_ITENS

CODIGO	DESCRICAO	TIPO	QUANTIDADE
1	'Biscoito'	'Comida'	20
2	'Água'	'Bebida'	47,95
3	'Suco'	'Bebida'	23,4
4	'Pão'	'Comida'	10
5	'Refrigerante'	'Bebida'	13,75

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_itens (  
  codigo INTEGER,  
  descricao VARCHAR2(200),  
  tipo VARCHAR(20),  
  quantidade DECIMAL(15,2));  
  
INSERT INTO ex_itens VALUES (1, 'Biscoito', 'Comida', 20);  
INSERT INTO ex_itens VALUES (2, 'Água', 'Bebida', 47.95);  
INSERT INTO ex_itens VALUES (3, 'Suco', 'Bebida', 23.4);  
INSERT INTO ex_itens VALUES (4, 'Pão', 'Comida', 10);  
INSERT INTO ex_itens VALUES (5, 'Refrigerante', 'Bebida', 13.75);
```

```
COMMIT;
```

## Consulta com agrupamento agregado

```
SELECT COUNT( QUANTIDADE) QTD_POR_TIPO,
       SUM( QUANTIDADE) VOLUME_POR_TIPO,
       MAX( DESCRICAO) KEEP ( DENSE_RANK FIRST ORDER BY QUANTIDADE)
ITEM_MENOR_QTD_POR_TIPO,
       MIN( QUANTIDADE) MENOR_QTD_POR_TIPO,
       MAX( DESCRICAO) KEEP ( DENSE_RANK LAST ORDER BY QUANTIDADE)
ITEM_MAIOR_QTD_POR_TIPO,
       MAX( QUANTIDADE) MAIOR_QTD_POR_TIPO,
       TRUNC( AVG( QUANTIDADE), 2) MEDIA_QTD_POR_TIPO,
       MEDIAN( QUANTIDADE) MEDIANA_QTD_POR_TIPO
FROM EX_ITENS
GROUP BY TIPO
```

### Retorno:

QTD_POR_TIPO	VOLUME_POR_TIPO	ITEM_MENOR_QTD_POR_TIPO	MENOR_QTD_POR_TIPO	ITEM_MAIOR_QTD_POR_TIPO	MAIOR_QTD_POR_TIPO	MEDIA_QTD_POR_TIPO	MEDIANA_QTD_POR_TIPO
3	85.1	'Refrigerante'	13.75	'Água'	47.95	28.36	23.4
2	30	'Pão'	10	'Biscoito'	20	15	15

- **Count:** Retorna quantidade de linhas no agrupamento definido;
- **Sum:** Retorna soma de uma coluna das linhas no agrupamento definido;
- **Max:** Retorna valor máximo do campo de entrada no agrupamento definido;
- **Min:** Retorna valor mínimo do campo de entrada no agrupamento definido;
- **First:** Retorna primeiro resultado pela ordenação definida, utilizando estrutura com **Keep** e **Dense\_rank**;
- **Last:** Retorna último resultado pela ordenação definida, utilizando estrutura com **Keep** e **Dense\_rank**;
- **Avg:** Retorna média dos valores do campo de entrada no agrupamento definido;
- **Median:** Retorna mediana dos valores do campo de entrada no agrupamento definido.

## Consulta com agrupamento analítico

```
SELECT CODIGO, DESCRICAO, TIPO, QUANTIDADE,
```

```

DENSE_RANK() OVER (ORDER BY QUANTIDADE DESC) RANK_QTD,
DENSE_RANK() OVER (PARTITION BY TIPO ORDER BY QUANTIDADE DESC)
RANK_QTD_POR_TIPO,
COUNT(QUANTIDADE) OVER (PARTITION BY TIPO) QTD_ITENS_POR_TIPO,
SUM(QUANTIDADE) OVER (PARTITION BY TIPO) QTD_VOLUME_POR_TIPO,
MAX(DESCRICAO) KEEP (DENSE_RANK FIRST ORDER BY QUANTIDADE) OVER (PARTITION BY TIPO) AS
ITEM_MENOR_QTD_POR_TIPO,
MIN(QUANTIDADE) OVER (PARTITION BY TIPO) MENOR_QTD_POR_TIPO,
MAX(DESCRICAO) KEEP (DENSE_RANK LAST ORDER BY QUANTIDADE) OVER (PARTITION BY TIPO) AS
ITEM_MAIOR_QTD_POR_TIPO,
MAX(QUANTIDADE) OVER (PARTITION BY TIPO) MAIOR_QTD_POR_TIPO,
TRUNC(AVG(QUANTIDADE) OVER (PARTITION BY TIPO), 2) MEDIA_QTD_POR_TIPO,
MEDIAN(QUANTIDADE) OVER (PARTITION BY TIPO) MEDIANA_QTD_POR_TIPO
FROM EX_ITENS

```

## Retorno:

CODIGO	DESCRICAO	TIPO	QUANTIDADE	RANK_QTD	RANK_QTD_POR_TIPO	QTD_ITENS_POR_TIPO	QTD_VOLUME_POR_TIPO	ITEM_MENOR_QTD_POR_TIPO	MENOR_QTD_POR_TIPO	ITEM_MAIOR_QTD_POR_TIPO	MAIOR_QTD_POR_TIPO	MEDIA_QTD_POR_TIPO	MEDIANA_QTD_POR_TIPO
2	'Água'	'Bebida'	47.95	1	1	3	85.1	'Refrigerante'	13.75	'Água'	47.95	28.36	23.4
3	'Suco'	'Bebida'	23.4	2	2	3	85.1	'Refrigerante'	13.75	'Água'	47.95	28.36	23.4
1	'Biscoito'	'Comida'	20	3	1	2	30	'Pão'	10	'Biscoito'	20	15	15
5	'Refrigerante'	'Bebida'	13.75	4	3	3	85.1	'Refrigerante'	13.75	'Água'	47.95	28.36	23.4
4	'Pão'	'Comida'	10	5	2	2	30	'Pão'	10	'Biscoito'	20	15	15

- **Dense\_rank():** Retorna posição de rank definida pela ordenação na estrutura com **Over**;
- **Partition by:** Utilizado para retornar um valor agrupado sem fazer um **distinct**.

# Nvl e Coalesce

Com as funções **NVL** e **COALESCE**, é possível definir que, quando o campo retornar nulo, outro valor será retornado no lugar.

## TABELA EXEMPLO

### EX\_PRODUTOS

IDPRODUTO	DESCRICAO	VALOR_CUSTO	VALOR_VENDA
1	'Arroz 1KG'	0.75	(null)
2	'Feijão 1KG'	1.07	2.99
3	'Macarrão 500G'	(null)	(null)

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_produtos (  
  idproduto INTEGER,  
  descricao VARCHAR2(200),  
  valor_custo DECIMAL(15,2),  
  valor_venda DECIMAL(15,2));  
INSERT INTO ex_produtos VALUES (1, 'Arroz 1KG', 0.75, NULL);  
INSERT INTO ex_produtos VALUES (2, 'Feijão 1KG', 1.07, 2.99);  
INSERT INTO ex_produtos VALUES (3, 'Macarrão 500G', NULL, NULL);
```

## NVL

A função NVL retorna o valor do segundo parâmetro quando o primeiro ser nulo. Só pode ser usado dois parâmetros de entrada.

```
SELECT descricao, NVL(valor_venda, valor_custo) AS valor FROM ex_produtos
```

### Retorno:

DESCRICAO	VALOR
'Arroz 1KG'	0.75
'Feijão 1KG'	2.99

```
| 'Macarrão 500G' | (null) |
```

## COALESCE

A função **COALESCE** retorna o valor do parâmetro seguinte quando o anterior ser nulo. Pode ser usado quantos parâmetros de entrada for necessário.

```
SELECT descricao, COALESCE(valor_venda, valor_custo, 0) AS valor FROM ex_produtos
```

### Retorno:

DESCRICAO	VALOR
'Arroz 1KG'	0.75
'Feijão 1KG'	2.99
'Macarrão 500G'	0



# Operações Join

Usamos as operações JOIN para relacionar dados de duas ou mais tabelas em uma consulta, utilizando igualdade de colunas em comum ou não.

Será mostrado um exemplo de situação onde pode ser usado os diferentes tipos de JOIN e seus resultados entre duas tabelas.

## TABELAS EXEMPLO

### EX\_ESTADO

	UF		DESCRICAO		-----		SC		Santa Catarina			SP		São Paulo	
--	----	--	-----------	--	-------	--	----	--	----------------	--	--	----	--	-----------	--

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_estado (uf VARCHAR2(2), descricao VARCHAR2(100));
INSERT INTO ex_estado VALUES ('SC', 'Santa Catarina');
INSERT INTO ex_estado VALUES ('SP', 'São Paulo');
COMMIT;
```

### EX\_CIDADE

	IDCIDADE		NOME		UF		-----		1		Criciúma		SC			2	
	Florianópolis		SC				3		Curitiba		PR						

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_cidade (idcidade INTEGER, nome VARCHAR2(200), uf VARCHAR2(2));
INSERT INTO ex_cidade VALUES (1, 'Criciúma', 'SC');
INSERT INTO ex_cidade VALUES (2, 'Florianópolis', 'SC');
INSERT INTO ex_cidade VALUES (3, 'Curitiba', 'PR');
COMMIT;
```

## INNER JOIN

[image-1646844726590.jpg](#)

Image not found or type unknown

É usado para relacionar e mostrar os dados de ambas as tabelas quando houver igualdade nos campos em comum.

```
SELECT ES. DESCRICAO AS ESTADO,
        CI. NOME AS CIDADE
FROM EX_ESTADO ES
INNER JOIN EX_CIDADE CI ON ES. UF = CI. UF
```

### Usos alternativos que retornam o mesmo resultado:

Não é necessário informar a operação **INNER**.

```
SELECT ES. DESCRICAO AS ESTADO,
        CI. NOME AS CIDADE
FROM EX_ESTADO ES
JOIN EX_CIDADE CI ON ES. UF = CI. UF
```

Utilizando com a operação **NATURAL**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES. DESCRICAO AS ESTADO,
        CI. NOME AS CIDADE
FROM EX_ESTADO ES NATURAL JOIN EX_CIDADE CI
```

Utilizando com a cláusula **USING**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES. DESCRICAO AS ESTADO,
        CI. NOME AS CIDADE
FROM EX_ESTADO ES
JOIN EX_CIDADE CI USING (UF)
```

Utilizando com a cláusula **WHERE**.

```
SELECT ES. DESCRICAO AS ESTADO,
        CI. NOME AS CIDADE
FROM EX_ESTADO ES, EX_CIDADE CI
WHERE ES. UF = CI. UF
```

### Retorno:

	ESTADO		CIDADE		-----		Santa Catarina		Criciúma			Santa
--	--------	--	--------	--	-------	--	----------------	--	----------	--	--	-------

## LEFT OUTER JOIN

image-1646845009495.jpg

Image not found or type unknown

É usado para relacionar e mostrar os dados de ambas as tabelas, preservando somente os dados da **primeira tabela** mesmo que não haja igualdade nos campos em comum.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
LEFT OUTER JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

### Usos alternativos que retornam o mesmo resultado:

Não é necessário informar a operação **OUTER**.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
LEFT JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

Utilizando com a operação **NATURAL**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES NATURAL LEFT JOIN EX_CIDADE CIB
```

Utilizando com a cláusula **USING**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
LEFT JOIN EX_CIDADE CI USING (UF)
```

### Retorno:

ESTADO	CIDADE		
Santa Catarina	Criciúma		
Santa			

## RIGHT OUTER JOIN

image-1646845256600.jpg

Image not found or type unknown

É usado para relacionar e mostrar os dados de ambas as tabelas, preservando somente os dados da **segunda tabela** mesmo que não haja igualdade nos campos em comum.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
RIGHT OUTER JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

### Usos alternativos que retornam o mesmo resultado:

Não é necessário informar a operação **OUTER**.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
RIGHT JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

Utilizando com a operação **NATURAL**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES NATURAL RIGHT JOIN EX_CIDADE CI
```

Utilizando com a cláusula **USING**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
RIGHT JOIN EX_CIDADE CI USING (UF)
```

### Retorno:

	ESTADO		CIDADE		-----		Santa Catarina		Florianópolis		
--	--------	--	--------	--	-------	--	----------------	--	---------------	--	--

## FULL OUTER JOIN

image-1646845502120.jpg

Image not found or type unknown

É usado para relacionar e mostrar os dados de ambas as tabelas, preservando os dados mesmo que não haja igualdade nos campos em comum.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
FULL OUTER JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

### Usos alternativos que retornam o mesmo resultado:

Não é necessário informar a operação **OUTER**.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
FULL JOIN EX_CIDADE CI ON ES.UF = CI.UF
```

Utilizando com a operação **NATURAL**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES NATURAL FULL JOIN EX_CIDADE CI
```

Utilizando com a cláusula **USING**. É necessário que as colunas em comum tenham o mesmo nome em ambas as tabelas

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
FULL JOIN EX_CIDADE CI USING (UF)
```

### Retorno:

	ESTADO		CIDADE		-----		Santa Catarina		Criciúma			Santa
--	--------	--	--------	--	-------	--	----------------	--	----------	--	--	-------

Catarina | Florianópolis | | (null) | Curitiba | | São Paulo | (null) |

## CROSS JOIN

É usado para mostrar os dados de ambas as tabelas sem relacionar por campos em comum. Essa operação relaciona todos registros de uma tabela com todos registros da outra tabela.

```
SELECT ES.DESCRICAO AS ESTADO,  
       CI.NOME AS CIDADE  
FROM EX_ESTADO ES  
CROSS JOIN EX_CIDADE CI
```

### Retorno:

```
| ESTADO | CIDADE | ----- | Santa Catarina | Criciúma | | Santa  
Catarina | Florianópolis | | Santa Catarina | Curitiba | | São Paulo | Criciúma | | São Paulo  
| Florianópolis | | São Paulo | Curitiba |
```

# Order by, Group by, Having e Distinct

Com os operadores **ORDER BY**, **GROUP BY**, **HAVING** e **DISTINCT** podemos organizar nossas consultas mais dinamicamente.

## TABELA EXEMPLO

### EX\_ITENS

CODIGO	DESCRICAO	TIPO	QUANTIDADE
1	'Biscoito'	'Comida'	20
2	'Água'	'Bebida'	47,95
3	'Suco'	'Bebida'	23,4
4	'Pão'	'Comida'	10
5	'Refrigerante'	'Bebida'	13,75

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_itens (  
  codigo INTEGER,  
  descricao VARCHAR2(200),  
  tipo VARCHAR(20),  
  quantidade DECIMAL(15,2));  
INSERT INTO ex_itens VALUES (1, 'Biscoito', 'Comida', 20);  
INSERT INTO ex_itens VALUES (2, 'Água', 'Bebida', 47.95);  
INSERT INTO ex_itens VALUES (3, 'Suco', 'Bebida', 23.4);  
INSERT INTO ex_itens VALUES (4, 'Pão', 'Comida', 10);  
INSERT INTO ex_itens VALUES (5, 'Refrigerante', 'Bebida', 13.75);  
COMMIT;
```

## ORDER BY

Ordenar  **crescente** pela coluna **TIPO**.

```
SELECT *
```

```
FROM EX_ITENS  
ORDER BY TIPO
```

### Retorno:

CODIGO	DESCRICAO	TIPO	QUANTIDADE
3	' Suco'	' Bebida'	23, 4
5	' Refrigerante'	' Bebida'	13, 75
2	' Água'	' Bebida'	47, 95
4	' Pão'	' Comida'	10
1	' Biscoito' ☐	' Comida'	20

Ordenar **crescente** pela coluna **TIPO** e **decrescente** pela coluna **QUANTIDADE**. (a ordenação respeita a sequência inserida)

```
SELECT *  
FROM EX_ITENS  
ORDER BY TIPO, QUANTIDADE DESC
```

### Retorno:

CODIGO	DESCRICAO	TIPO	QUANTIDADE
2	' Água'	' Bebida'	47, 95
3	' Suco'	' Bebida'	23, 4
5	' Refrigerante'	' Bebida'	13, 75
1	' Biscoito' ☐	' Comida'	20
4	' Pão'	' Comida'	10

## GROUP BY

Agrupar dados pela coluna **TIPO** contando a quantidade de **itens** (linhas) e **volumes** (coluna **QUANTIDADE**) para cada tipo de item.

```
SELECT TIPO,  
COUNT(1) AS QTD_ITEM,  
SUM( QUANTIDADE) AS QTD_VOLUME  
FROM EX_ITENS GROUP BY TIPO
```

### Retorno:



TIPO	QTD_ITEM	QTD_VOLUME
'Bebida'	3	85,1
'Comida'	2	30

## HAVING

Agrupar pela coluna **TIPO** mostrando somente quando a **soma** da coluna **QUANTIDADE** é **menor do que 40**.

```
SELECT TIPO, SUM( QUANTIDADE) AS QTD_VOLUME
FROM EX_ITENS
GROUP BY TIPO HAVING SUM( QUANTIDADE) < 40
```

**Retorno:**

TIPO	QTD_VOLUME
'Comida'	30

Agrupar pela coluna **TIPO** mostrando somente quando a contagens de linhas é **maior ou igual à 3**.

```
SELECT TIPO, COUNT(1) AS QTD_ITEM
FROM EX_ITENS
GROUP BY TIPO HAVING COUNT(1) >= 3
```

**Retorno:**

TIPO	QTD_ITEM
'Bebida'	3

## DISTINCT

É uma clausula SQL a qual pega os valores da tabelas e traz somente os valores "diferentes". Caso haja valores duplicados, com esta função o retorno é apenas um valor.

Sem o DISTINCT:

```
SELECT TIPO
FROM EX_ITENS
```

```
WHERE TIPO LIKE '%Comida%'
```

### Retorno:

```
| TIPO      |  
-----  
| 'Comida'  |  
| 'Comida'  |
```

Com o DISTINCT:

```
SELECT DISTINCT TIPO  
FROM EX_ITENS  
WHERE TIPO LIKE '%Comida%'
```

**Retorno:** Ou seja, caso houver mais de um valor correspondente com o filtro, o distinct apenas traz um, sem duplicá-los na consulta.

```
| TIPO      |  
-----  
| 'Comida'  |
```

Em questão de performance entre **DISTINCT** ou **GROUP BY** são iguais. Entretanto o uso do **GROUP BY** são mais usados para agrupar valores de funções analítica tais como **SUM, AVG, COUNT, ETC...**

# Pacote de Funções DOX (Package) - Oracle

Pensando em facilitar a busca de informações dentro do software por meio de instruções SQL, ao longo do tempo foram criadas várias funções e procedures que simplificaram ainda mais a busca de informações, seja por nossos consultores internos, ou por nossos clientes em suas automações.

A Ema disponibiliza um pacote para criar no banco de dados que executará todas as funções e procedures instantaneamente e de uma só vez.

Para isso, basta executar os scripts abaixo para o banco especificado:

- 

## **Primeiramente, executar a criação da package**

```
create or replace package pkg_ema as
  --variaveis
  function retorna_valor_variavel(xidprocesso in integer, xidvariavel in integer) return
varchar2;
  function retorna_valor_variavel_texto(xidprocesso in integer, xidvariavel in integer)
return varchar2;
  function retorna_valor_variavel_int(xidprocesso in integer, xidvariavel in integer) return
number;
  function retorna_valor_variavel_valor(xidprocesso in integer, xidvariavel in integer)
return number;
  function retorna_valor_variavel_data(xidprocesso in integer, xidvariavel in integer)
return date;
  function retorna_valor_variavel_datahr(xidprocesso in integer, xidvariavel in integer)
return date;

  --grade de dados
  function retorna_coluna_grade(xidprocesso in number, xidatividade in number, xidformulario
in number, xidgrade in number, xidvalor in number) return varchar;
  function retorna_coluna_grade_texto(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return varchar;
  function retorna_coluna_grade_int(xidprocesso in number, xidatividade in number,
```

```

xidformulario in number, xidgrade in number, xidvalor in number) return number;

function retorna_coluna_grade_valor(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return number;

function retorna_coluna_grade_data(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return date;

function retorna_coluna_grade_datahr(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return date;

end;

```

- 

## Em seguida, executar a criação do body

```

create or replace package body pkg_ema as
-- variaveis

function retorna_valor_variavel(xidprocesso in integer, xidvariavel in integer) return
varchar2 is valor varchar2(5000);
begin
    select (select valoratual from dual) into valor
    from crm_processo_variavel
    where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
end;

function retorna_valor_variavel_texto(xidprocesso in integer, xidvariavel in integer)
return varchar2 is valor varchar2(5000);
begin
    select (select valoratual from dual) into valor
    from crm_processo_variavel
    where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
end;

function retorna_valor_variavel_int(xidprocesso in integer, xidvariavel in integer) return
number is valor number(15,2);
begin
    select (select to_number(decode(valoratual, '.', 0, valoratual)) from dual)
    into valor
    from crm_processo_variavel

```

```

        where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
end;

function retorna_valor_variavel_valor(xidprocesso in integer, xidvariavel in integer)
return number is valor number(15,2);
begin
    select replace(replace(decode(upper(valoratual), '.', '0', upper(valoratual)) , '.', ''),
    ',', '.'))
        into valor
        from crm_processo_variavel
        where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
end;

function retorna_valor_variavel_data(xidprocesso in integer, xidvariavel in integer)
return date is valor date;
begin
    if (xidprocesso > 0 and xidvariavel > 0) then
        begin
            select case when valoratual <> '.' then to_date(valoratual, 'dd/mm/yyyy
HH24:MI')
                                else to_date('01-01-1900', 'dd/mm/yyyy HH24:MI') end
        into valor
        from crm_processo_variavel
        where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
    end;
    else
        valor := to_date('01/01/1900', 'dd/mm/yyyy');
    end if;
    return trunc(valor);
end;

function retorna_valor_variavel_datahr(xidprocesso in integer, xidvariavel in integer)
return date is valor date;
begin
    if (xidprocesso > 0 and xidvariavel > 0) then

```

```

begin
    select case when valoratual <> '.' then to_date(valoratual,'dd/mm/yyyy
HH24: MI' )
                                else to_date('01-01-1900','dd/mm/yyyy HH24: MI' ) end
into valor
    from crm_processo_variavel
    where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
end;
else
    valor := to_date('01/01/1900','dd/mm/yyyy HH24: MI');
end if;
return valor;
end;

--grade de dados
function retorna_coluna_grade(xidprocesso in number, xidatividade in number,  xidformulario
in number, xidgrade in number, xidvalor in number) return varchar is leitura varchar(4000);
begin
    select resposta into leitura
    from crm_processo_grade_valor
    where idprocesso    = xidprocesso and
        idformulario = xidformulario and
        idatividade   = xidatividade and
        idgrade       = xidgrade and
        idvalor       = xidvalor and
        idrepeticao    = (select max(idrepeticao)
                        from crm_processo_grade_valor xx
                        where xx.idatividade =
crm_processo_grade_valor.idatividade
                        and xx.idprocesso   =
crm_processo_grade_valor.idprocesso
                        and xx.idformulario =
crm_processo_grade_valor.idformulario
                        and xx.idgrade      = crm_processo_grade_valor.idgrade);

    return( trim( leitura));
end;

function retorna_coluna_grade_texto(xidprocesso in number, xidatividade in number,

```

```
xidformulario in number, xidgrade in number, xidvalor in number) return varchar is leitura  
varchar( 4000);
```

```
begin  
    select resposta into leitura  
        from crm_processo_grade_valor  
    where idprocesso    = xidprocesso and  
        idformulario   = xidformulario and  
        idatividade    = xidatividade and  
        idgrade        = xidgrade and  
        idvalor        = xidvalor and  
        idrepeticao     = (select max(idrepeticao)  
                           from crm_processo_grade_valor xx  
                           where xx.idatividade =  
crm_processo_grade_valor.idatividade  
                           and xx.idprocesso    =  
crm_processo_grade_valor.idprocesso  
                           and xx.idformulario =  
crm_processo_grade_valor.idformulario  
                           and xx.idgrade      = crm_processo_grade_valor.idgrade);
```

```
    return( trim( leitura));
```

```
end;
```

```
function retorna_coluna_grade_int(xidprocesso in number, xidatividade in number,  
xidformulario in number, xidgrade in number, xidvalor in number) return number is leitura  
number(15,2);
```

```
begin
```

```
    leitura := 0 ;
```

```
    select (select to_number(decode(respostainteiro, '.', 0, respostainteiro)) from dual) into  
leitura
```

```
        from crm_processo_grade_valor
```

```
    where idprocesso    = xidprocesso and
```

```
        idformulario   = xidformulario and
```

```
        idatividade    = xidatividade and
```

```
        idgrade        = xidgrade and
```

```
        idvalor        = xidvalor and
```

```
        idrepeticao     = (select max(idrepeticao)
```

```
                           from crm_processo_grade_valor xx
```

```
                           where xx.idatividade =
```

```

crm_processo_grade_valor.idatividade
                                and xx.idprocesso    =
crm_processo_grade_valor.idprocesso
                                and xx.idformulario =
crm_processo_grade_valor.idformulario
                                and xx.idgrade       = crm_processo_grade_valor.idgrade);

    return( leitura );
end;

function retorna_coluna_grade_valor(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return number is leitura
number(15,2);
begin
    leitura := 0;
    select respostavalor into leitura
    from crm_processo_grade_valor
    where idprocesso = xidprocesso and
          idformulario = xidformulario and
          idatividade = xidatividade and
          idgrade      = xidgrade and
          idvalor       = xidvalor and
          idrepeticao    = (select max(idrepeticao)
                           from crm_processo_grade_valor xx
                           where xx.idatividade =
crm_processo_grade_valor.idatividade
                                and xx.idprocesso    =
crm_processo_grade_valor.idprocesso
                                and xx.idformulario =
crm_processo_grade_valor.idformulario
                                and xx.idgrade       = crm_processo_grade_valor.idgrade);

    return(leitura);
end;

function retorna_coluna_grade_data(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return date is leitura
date;
begin
    select trunc(respostadatahora) into leitura
    from crm_processo_grade_valor
    where idprocesso = xidprocesso and

```



```

        idformulario = xidformulario and
        idatividade = xidatividade and
        idgrade      = xidgrade and
        idvalor       = xidvalor and
        idrepeticao    = (select max(idrepeticao)
                        from crm_processo_grade_valor xx
                        where xx.idatividade =
crm_processo_grade_valor.idatividade
                        and xx.idprocesso =
crm_processo_grade_valor.idprocesso
                        and xx.idformulario =
crm_processo_grade_valor.idformulario);
        return( leitura);
    end;

function retorna_coluna_grade_datahr(xidprocesso in number, xidatividade in number,
xidformulario in number, xidgrade in number, xidvalor in number) return date is leitura
date;
begin
    select respostadatahora into leitura
    from crm_processo_grade_valor
    where idprocesso = xidprocesso and
        idformulario = xidformulario and
        idatividade = xidatividade and
        idgrade      = xidgrade and
        idvalor       = xidvalor and
        idrepeticao    = (select max(idrepeticao)
                        from crm_processo_grade_valor xx
                        where xx.idatividade =
crm_processo_grade_valor.idatividade
                        and xx.idprocesso =
crm_processo_grade_valor.idprocesso
                        and xx.idformulario =
crm_processo_grade_valor.idformulario);
    return( leitura);
end;
end;

```

## Chamando as Funções

- **Variável:**

```
select pkg_ema.retorna_valor_variavel(idprocesso, 1) as padrao,  
       pkg_ema.retorna_valor_variavel_texto(idprocesso, 2) as texto,  
       pkg_ema.retorna_valor_variavel_int(idprocesso, 3) as inteiro,  
       pkg_ema.retorna_valor_variavel_valor(idprocesso, 4) as valor,  
       pkg_ema.retorna_valor_variavel_data(idprocesso, 5) as data,  
       pkg_ema.retorna_valor_variavel_datahr(idprocesso, 6) as datahr  
from crm_processo where idprocesso = 0/*IDPROCESSO*/
```

- **Grade:**

```
select pkg_ema.retorna_coluna_grade(x.idprocesso, x.idatividade, x.idformulario, 1,  
x.idvalor) as padrao,  
       pkg_ema.retorna_coluna_grade_texto(x.idprocesso, x.idatividade, x.idformulario, 2,  
x.idvalor) as texto,  
       pkg_ema.retorna_coluna_grade_int(x.idprocesso, x.idatividade, x.idformulario, 3,  
x.idvalor) as inteiro,  
       pkg_ema.retorna_coluna_grade_valor(x.idprocesso, x.idatividade, x.idformulario, 4,  
x.idvalor) as valor,  
       pkg_ema.retorna_coluna_grade_data(x.idprocesso, x.idatividade, x.idformulario, 5,  
x.idvalor) as data,  
       pkg_ema.retorna_coluna_grade_datahr(x.idprocesso, x.idatividade, x.idformulario, 6,  
x.idvalor) as datahr  
from crm_processo_grade_valor x  
where idprocesso = 0/*IDPROCESSO*/ and idatividade = 0/*IDATIVIDADE*/ and idformulario =  
0/*IDFORMULARIO*/  
and idrepeticao = (select max(idrepeticao) from crm_processo_grade_valor where idprocesso  
= x.idprocesso and idatividade = x.idatividade)  
and idgrade = 1 order by idvalor
```

# Retorna datas de um intervalo de dias

## SQL:

```
select x.data,
       initcap(to_char(x.data,' DAY' )) dia_semana,
       case when to_char(x.data, 'd') in (1,7) then 'Fim de semana' else 'Dia útil' end
legenda
from (select data_inicial + level - 1 data
      from (select to_date('12/07/2019', 'DD/MM/YYYY') data_inicial from dual)
      connect by level <= sysdate - data_inicial + 1) x
order by x.data
```

## Retorno:

DATA	DIA_SEMANA	LEGENDA
12/07/2019	Sexta-Feira	Dia útil
13/07/2019	Sábado	Fim de semana
14/07/2019	Domingo	Fim de semana
15/07/2019	Segunda-Feira	Dia útil
16/07/2019	Terça-Feira	Dia útil
17/07/2019	Quarta-Feira	Dia útil

# Retorna tempo de um intervalo de datas

SQL:

```

select --campos individuais
       trunc((datafim-datainicio) / 365) totalano,
       trunc((datafim-datainicio) / 30) totalmes,
       trunc((datafim-datainicio)) totaldia,
       trunc(((datafim-datainicio) * 24)) totalhora,
       trunc(((datafim-datainicio) * 1440)) totalminuto,
       --campo consolidado por extenso
       to_char(trunc((datafim-datainicio) / 365)) ||
         case when trunc((datafim-datainicio) / 365) <> 1 then ' anos, ' else ' ano, ' end ||
       to_char(trunc((datafim-datainicio) / 30) - trunc((datafim-datainicio) / 365) * 12) ||
         case when (trunc((datafim-datainicio) / 30) - trunc((datafim-datainicio) / 365) *
12) <> 1 then ' meses, ' else ' mês, ' end ||
       to_char(trunc((datafim-datainicio)) - trunc((datafim-datainicio) / 30) * 30) ||
         case when (trunc((datafim-datainicio)) - trunc((datafim-datainicio) / 30) * 30) <> 1
then ' dias, ' else ' dia, ' end ||
       to_char(trunc(((datafim-datainicio) * 24)) - trunc((datafim-datainicio)) * 24) ||
         case when (trunc(((datafim-datainicio) * 24)) - trunc((datafim-datainicio)) * 24) <>
1 then ' horas e ' else ' hora e ' end ||
       to_char(trunc(((datafim-datainicio) * 1440)) - trunc(((datafim-datainicio) * 24)) *
60) ||
         case when (trunc(((datafim-datainicio) * 1440)) - trunc(((datafim-datainicio) * 24))
* 60) <> 1 then ' minutos' else ' minuto' end extenso
       --atribuir data início e data fim
       from (select to_date('01/01/2018 00:00', 'dd/mm/yyyy hh24:mi') datainicio,
                    to_date('22/07/2019 00:42', 'dd/mm/yyyy hh24:mi') datafim
              from dual)

```

## Retorno:

EXTENSO	TOTALANO	TOTALMES	TOTALDIA	TOTALHORA	TOTALMINUTO
1 ano, 6 meses, 27 dias, 0 horas e 42 minutos	1	18	567	13608	816522

# SQL - Descobrir se existem registros duplicados

Neste conteúdo vamos mostrar como verificar se existe registros duplicados no banco. Neste exemplo vamos mostrar como fazer na tabela CRM\_PROCEDIMENTO\_FORMULARIOS, porém o mesmo pode ser adaptado para qualquer outra tabela.

```
select cpa.idprocedimento,
       cpa.idatividade,
       cpa.idversao,
       idformulario,
       count(cpa.idprocedimento)
from   crm_procedimento_formulario cpa
group by  cpa.idprocedimento,
          cpa.idatividade,
          cpa.idversao, idformulario
having   count(cpa.idprocedimento) >=2
```

Retorno da consulta:

[image-1640005153780.png](#)

Image not found or type unknown

O count mostra quantos registro tem idênticos. Agora para descobrir quais registro são basta executar o seguinte SQL:

```
select *
from (select *
      from crm_procedimento_formulario
      where idprocedimento = 24
      and idatividade = 2
      and idversao = 1
      and idformulario = 1)
where rownum < 3
```

**Obs:** o comando rownum varia conforme a quantidade for retornada. Neste exemplo como os

filtros que colocamos ele retorno que havia 3 registros iguais o rownum fica " < 3" .

Retorno da consulta:

[image-1640005294965.png](#)

Image not found or type unknown

# SQL - Usuários com licenças do DOX Portal

Neste conteúdo será mostrado como listar via banco (SQL) os usuários que estão com licença ativa no DOX Portal para que possa fazer um controle sobre as licenças e saber quem está usando.

SQL:

```
select u.idusuario,
       u.username as usuario,
       case cfe.tipocliforemp
         when 0 then 'CLIENTE'
         when 1 then 'FORNECEDOR'
         when 2 then 'COLABORADOR' end as tipo
from usuario u
join usuario_programa up on u.idusuario = up.idusuario
join cliforemp cfe on cfe.idcliforemp = u.idcliforemp
where up.idtipoprograma = 18
and u.username <> 'GERENTE'
and u.inativo = 'N'
and (replace('/*TIPO*/', 'TIPO', '.') = '/*.**/'
or cfe.tipocliforemp = case '/*TIPO*/'
                        when 'CLIENTE' then 0
                        when 'FORNECEDOR' then 1
                        when 'COLABORADOR' then 2 end)
order by tipo desc, usuario
```

Retorno da consulta:

[image-1640006826481.png](#)

Image not found or type unknown



# Union, Union All, Minus e Intersect

É possível combinar os resultados de duas ou mais consultas através dos operadores **Union**, **Minus** e **Intersect**.

Será mostrado um exemplo de situação onde esses operadores podem ser usados e os resultados entre duas tabelas.

## TABELAS EXEMPLO

### EX\_FILIAL

DESCRICAO
'Matriz'
'Filial 1'
'Filial 2'
'Filial 3'

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_filial (descricao VARCHAR2(20));  
INSERT INTO ex_filial VALUES ('Matriz');  
INSERT INTO ex_filial VALUES ('Filial 1');  
INSERT INTO ex_filial VALUES ('Filial 2');  
INSERT INTO ex_filial VALUES ('Filial 3');  
COMMIT;
```

### EX\_PONTOS\_DE\_VENDA

DESCRICAO
'Filial 1'
'Filial 2'
'Filial 3'
'Unidade 1'

```
| 'Unidade 2' |
```

## SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_pontos_de_venda (descricao VARCHAR2(20));
INSERT INTO ex_pontos_de_venda VALUES ('Filial 1');
INSERT INTO ex_pontos_de_venda VALUES ('Filial 2');
INSERT INTO ex_pontos_de_venda VALUES ('Filial 3');
INSERT INTO ex_pontos_de_venda VALUES ('Unidade 1');
INSERT INTO ex_pontos_de_venda VALUES ('Unidade 2');
COMMIT;
```

## UNION

O operador UNION une o retorno de duas consultas fazendo DISTINCT.

```
SELECT DESCRICA0
  FROM EX_FILIAL
UNION
SELECT DESCRICA0
  FROM EX_PONTOS_DE_VENDA
```

### Retorno:

```
| DESCRICA0 |
-----
| 'Filial 1' |
| 'Filial 2' |
| 'Filial 3' |
| 'Matriz'   |
| 'Unidade 1'|
| 'Unidade 2'|
```

## UNION ALL

O operador **UNION ALL** une o retorno de duas consultas sem fazer **DISTINCT**.

```
SELECT DESCRICA0
  FROM EX_FILIAL
UNION ALL
SELECT DESCRICA0
```

```
FROM EX_PONTOS_DE_VENDA
```

### Retorno:

```
| DESCRICAO |  
-----  
| 'Matriz'  |  
| 'Filial 1'|  
| 'Filial 2'|  
| 'Filial 3'|  
| 'Filial 1'|  
| 'Filial 2'|  
| 'Filial 3'|  
| 'Unidade 1'|  
| 'Unidade 2'|
```

## MINUS

O operador **MINUS** não mostra qualquer dado da primeira consulta que se repete na segunda.

```
SELECT DESCRICAO  
FROM EX_FILIAL  
MINUS  
SELECT DESCRICAO  
FROM EX_PONTOS_DE_VENDA
```

### Retorno:

```
| DESCRICAO |  
-----  
| 'Matriz'  |
```

## INTERSECT

O operador **INTERSECT** mostra somente os dados que contem em ambas as consultas.

```
SELECT DESCRICAO
      FROM EX_FILIAL
INTERSECT
SELECT DESCRICAO
      FROM EX_PONTOS_DE_VENDA
```

**Retorno:**

```
| DESCRICAO |
-----
| 'Filial 1' |
| 'Filial 2' |
| 'Filial 3' |
```

# Utilização de Datas e Conversões (current\_date, current\_time, timestamp, etc)

## CURRENT\_DATE

Seleciona data atual (função alternativa: sysdate para pegar a data do sistema/servidor)

```
select current_date from dual --> Retorno: "2019-09-13"
```

## CURRENT\_TIMESTAMP

Seleciona data e hora atual (função alternativa: **systimestamp** para pegar a data e hora do sistema/servidor)

```
select current_timestamp from dual --> Retorno: "2019-09-13 08:56:29.649098-03"
```

**\*\*\*OBSERVAÇÃO:** Importante saber que o banco estará parametrizado para mostrar um valor de data/hora com um único formato para todos. Se a intenção é mostrar a data/hora em um formato diferente do parametrizado, é necessário converter a data/hora para um valor do tipo texto, definindo o formato desejado nas funções de conversão. Veja abaixo.

## TO\_CHAR

Pode ser usado para converter uma data/hora para texto no formato em que deseja visualizar.

```
select to_char(current_date, 'MM/YYYY') from dual --> Retorno: "09/2019"
```

```
select to_char(current_date, 'DD/MM/YYYY HH24:MI:SS') from dual --> Retorno: "13/09/2019 08:56:29"
```

```
select to_char(current_date, 'HH24:MI') from dual --> Retorno: "08:56"
```

```
select to_char(current_date, 'Day') from dual --> Retorno: "Sexta-Feira"
```

```
select to_char(current_date, 'DD')||' de '||trim(to_char(current_date, 'Month'))||' de '||to_char(current_date, 'YYYY') from dual --> Retorno: "13 de Setembro de 2019"
```

**\*\*\*OBSERVAÇÃO:** Em Oracle, a função `current_date` também é usada para selecionar horas, minutos e segundos.

## TO\_DATE E TO\_TIMESTAMP

Usado para converter um valor texto para um valor data/hora.

```
select to_date('13/09/2019', 'DD/MM/YYYY') from dual --> Retorno: "2019-09-13"
```

```
select to_timestamp('13/09/2019 08:51:43', 'DD/MM/YYYY HH24:MI:SS') from dual --> Retorno: "2019-09-13 08:51:43-03"
```

## EXTRACT

Pode ser usado para extrair uma informação específica de um campo de Data ou Data/hora.

```
select extract(year from sysdate) from dual --> Retorno: 2019 <-- (ano atual)
```

```
select extract(month from sysdate) from dual --> Retorno: 9 <-- (mês atual)
```

```
select extract(day from sysdate) from dual --> Retorno: 13 <-- (dia do mês atual)
```

```
select extract(hour from sysdate)||':'||  
□ extract(minute from sysdate)||':'||  
□ trunc(extract(second from sysdate)) from dual --> Retorno: 14:43:46 <-- (hora atual)
```

## CAST

Função para fazer conversão de valores de diversos tipos. Neste caso, de texto para data (A data precisa estar informada no mesmo formato parametrizado no banco de dados).

```
select cast('13-09-2019' as date) from dual --> Retorno: "2019-09-13"
```

# With, Substr e Instr

Neste exemplo será mostrado como utilizar as funções **With**, **Substr** e **Instr** combinadas. Vamos dividir em colunas, as informações de uma nota fiscal parametrizadas por posição.

## Texto com as informações:

```
' NF 123456 - DT 07/10/2019 - VL 37,99'
```

## Código:

```
WITH nota AS (SELECT ' NF 123456 - DT 07/10/2019 - VL 37,99' AS dados FROM dual)
SELECT SUBSTR(dados, INSTR(dados, ' NF' )+3, INSTR(dados, ' - ' )-(INSTR(dados, ' NF' )+4)) AS
numeronf,
       SUBSTR(dados, INSTR(dados, ' DT' )+3, INSTR(dados, ' - ', 1, 2)-(INSTR(dados, ' DT' )+4)) AS
dataemissao,
       SUBSTR(dados, INSTR(dados, ' VL' )+3) AS valortotal
FROM nota
```

## Retorno:

NUMERONF	DATAEMISSAO	VALORTOTAL
-----		
123456	07/10/2019	37,99

## Neste caso:

- **With:** está sendo utilizado para que não seja preciso retornar o texto em todas as colunas, evitando repetir código e tempo de processamento;
- **Substr:** está retornando dados a partir de uma posição até outra informadas do texto;
- **Instr:** retorna a posição do texto, onde se encontra o caractere ou conjunto de caracteres informado, para usar a posição no **Substr**;