

# PostgreSQL

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

- [BACKUP - Resolução de problema com o pg\\_dump](#)
- [Coalesce](#)
- [Converter Binary Data \(Bytea\) para Text](#)
- [Criação de View](#)
- [Descobrir a maior versão de um procedimento](#)
- [Descobrir SQL Executado](#)
- [Erros de SQL comuns no banco PostgreSQL](#)
- [Exportando consulta do banco de dados \(Excel, TXT\)](#)
- [Exportando tabela em CSV \(Excel\)](#)
- [Função - Formata CNPJ](#)
- [Função - Formata CPF](#)
- [Função - Primeiro e ultimo dia do mês](#)
- [Função - Retorna parte de um campo de texto definindo inicio e fim](#)
- [Função de agregação \(count, sum, max, min, avg, partition by\)](#)
- [Função de Strings](#)
- [Funções de data](#)
- [Gerar log dos comandos](#)
- [Operações Join](#)
- [Order by, Group by e Having](#)
- [Pacote de Funções DOX \(Schema\) - PostgreSQL](#)
- [Resultados do SELECT separados por virgula](#)
- [Retorna lista](#)
- [Tamanhos da tabelas \(MB\)](#)
- [Union, Union All, Except, Except All e Intersect](#)
- [Utilização de Datas e Conversões \(current\\_date, timestamp, to\\_char, etc\)](#)
- [With, Substr, Split\\_part e Strpos](#)

# BACKUP - Resolução de problema com o pg\_dump

Ao tentar importar uma base e o diretório retornar erro no pg\_dump. A resolução seguirá abaixo:

Em Barra inicial> Pesquisar por "Configurações avançadas do sistema"> Aba "avançado"> Clicar em "variáveis de ambiente" > editar path> novo> setar caminho do pg\_dump (está dentro do disco local> arquivo de programa> seleciona arquivo do postgresql> seleciona pg\_dump.hba). Após isso é só finalizar a operação com "ok".

Conforme seguirá em imagens:

image-1693337301662.png

Image not found or type unknown

image-1693337323898.png

Image not found or type unknown

image-1693337354668.png

Image not found or type unknown

image-1693337487018.png

Image not found or type unknown

image-1693337503177.png

Image not found or type unknown

image-1693337441277.png

Image not found or type unknown

image-1693337542250.png

Image not found or type unknown

image-1693337555681.png

Image not found or type unknown

# Coalesce

Com a função **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 INT4,  
  descricao TEXT,  
  valor_custo DECIMAL(15,2),  
  valor_venda DECIMAL(15,2));  
INSERT INTO ex_produtos VALUES (1, 'Arroz 1KG', 0.75, NULL),  
                                (2, 'Feijão 1KG', 1.07, 2.99),  
                                (3, 'Macarrão 500G', NULL, 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  
  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	

# Converter Binary Data (Bytea) para Text

```
select convert_from(CAMPO_BINARY, 'UTF-8') from TABELA
```

# 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



# Descobrir a maior versão de um procedimento

Olá, compartilhamos aqui com vocês um SQL que traz a informação de procedimento e sua versão mais recente, levando em conta se o processo está ativo. Caso necessário, podes personalizar o SQL para trazer outras informações:

```
SELECT IDPROCEDIMENTO,  
       DESCRICAO,  
       IDVERSAO  
FROM CRM_PROCEDIMENTO X  
WHERE IDVERSAO = ( SELECT MAX ( IDVERSAO)  
                   FROM CRM_PROCEDIMENTO  
                   WHERE IDPROCEDIMENTO = X.IDPROCEDIMENTO)  
  
AND INATIVO = ' N'  
ORDER BY IDPROCEDIMENTO
```

# Descobrir SQL Executado

Abaixo comando util para monitorar / descobrir SQLs executados, muito util para descobrir SQLs para relatórios, realizar debug de SQL montados durante um processo ou para demais necessidades:

```
SELECT USERNAME,  
       APPLICATION_NAME,  
       QUERY_START,  
       QUERY  
FROM PG_STAT_ACTIVITY;
```

Foi montado uma visualização de exemplo, trazendo o usuário (Sim, o nome da coluna é USERNAME, não esta errado), a aplicação que esta usando, a data hora que rodou e o SQL, caso queira montar uma visualização personalizada, executar o comando abaixo e verificar quais colunas são mais uteis para sua necessidade.

```
SELECT *  
FROM PG_STAT_ACTIVITY;
```

Consultando Query ativas / lentas que estão afetando o desempenho do banco de dados.

```
SELECT *  
FROM PG_STAT_ACTIVITY  
WHERE UPPER( DATNAME ) = UPPER( ' ema' )  
AND UPPER( STATE ) = UPPER( ' active' );
```

# Erros de SQL comuns no banco PostgreSQL

## Invalid input syntax for integer

Esse erro ocorre quando se tenta dar um CAST em um campo inteiro com um valor inválido.

```
SELECT CAST(' 123x' AS INTEGER)
FROM VERSAODB
```

### SOLUÇÃO abaixo:

```
SELECT CAST(' 123' AS INTEGER)
FROM VERSAODB
```

## UNION types character varying and integer cannot be matched

Isso ocorrerá quando se está utilizando um comando UNION e os tipos de campos das colunas selecionadas são diferentes.

```
SELECT P. DOCUMENTO
FROM PAGAR P

UNION ALL

SELECT CP. IDDOC
FROM CHEQUE_PRE CP
```

### SOLUÇÃO abaixo: A solução é sempre igualar os tipos.

```
SELECT P. DOCUMENTO
FROM PAGAR P
```

```
UNION ALL
```

```
SELECT CAST( CP. IDDOC AS VARCHAR)  
FROM CHEQUE_PRE CP
```

**current transaction is aborted, commands ignored until end of transaction block**

Isso começa a ocorrer quando você está em uma sessão com o AUTO COMMIT desabilitado onde você executa um SQL Statement (SELECT \* FROM...) e o mesmo tem um na sua sintaxe.

Quando isso ocorre o PostgreSQL aborta e não termina a execução do comando e sendo necessário fazer um ROLLBACK. Quando o AUTO COMMIT está habilitado o ROLLBACK é automático. Porém os COMMIT também.

**date/time field value out of range: "05/31/2019 00:00:00"**

Este erro acontece quando é tentado filtrar uma data/hora em um campo que o tipo dele é somente data.

```
SELECT *  
FROM NFS  
WHERE NFS. DATAEMISSAO BETWEEN ' 05/31/2019 00: 00: 00' AND ' 05/31/2021 00: 00: 00'
```

**SOLUÇÃO abaixo:**

```
SELECT *  
FROM NFS  
WHERE NFS. DATAEMISSAO BETWEEN ' 2019-05-31' AND ' 2019-05-31'
```

# Exportando consulta do banco de dados (Excel, TXT)

Neste fórum vamos mostrar como exportar sua consulta SQL para um arquivo CSV ou um TXT gerado em sua máquina. Ex: Utilize o seguinte comando para criar o arquivo:

```
COPY
(  SELECT C. IDCLIFOREMP AS IDCLIFOREMP,
      C. FANTASIA AS NOME,
      U. USERNAME AS USERNAME,
      C. EMAIL AS EMAIL,
      D. DESCRICAO AS DEPARTAMENTO,
      STRING_AGG( BF. DESCRICAO, ' , ' )
FROM CLIFOREMP C
      JOIN USUARIO U ON ( C. IDCLIFOREMP = U. IDCLIFOREMP)
[] JOIN USUARIO_PAPELFUNCAO UP ON ( U. IDUSUARIO = UP. IDUSUARIO)
[] JOIN BPM_FUNCAO BF ON ( BF. IDBPMFUNCAO = UP. IDPAPELFUNCAO)
[] JOIN DEPARTAMENTOS D ON ( C. IDDEPARTAMENTO = D. IDDEPARTAMENTO)
[] WHERE U. INATIVO = ' N'
[] AND C. INATIVO = ' N'
[] GROUP BY C. IDCLIFOREMP,
[]      C. FANTASIA,
[]      U. USERNAME,
[]      C. EMAIL,
[]      D. DESCRICAO)

TO 'C: /temp/teste.csv'
DELIMITER ';'
CSV HEADER
```

**COPY** — Comando que é utilizado para a exportação do arquivo.

**( )** — Dentro do parênteses, abaixo do comando, é onde estará o SQL que irá trazer os dados para a geração do arquivo. Como exemplo, fiz um SQL que traga todos os papéis funções de todos os usuários do DOX.

**TO 'C:/temp/teste.csv'** — Local onde o arquivo será gerado e a extensão. Caso queira trocar para um TXT por exemplo, apenas mudar o nome da extensão.

**DELIMITER ';'**

— Após o “Delimiter” deverá ser colocado o separador desejado. O separador pode ser ";", ",", "|", etc... neste caso, foi utilizado o “;” como exemplo.

**CSV HEADER** — Utiliza-se esse comando caso queira que o nome das colunas apareçam no arquivo gerado(cabeçalho).

**Para que o arquivo seja criado, sempre verificar as permissões das pastas do diretório.**

image-1646770252576.png

Image not found or type unknown

Esse comando de exportar a consulta pode ser usado em um processo, por exemplo, utilizando o evento **Registro banco dados - SQL - Executar**, com a opção “comando” marcado no conector.

image-1646770271448.png

Image not found or type unknown

# Exportando tabela em CSV (Excel)

Estamos compartilhando com todos uma maneira pratica de exportar uma tabela inteira do banco de dados PostgreSQL para o formato CSV para ser manipulado fora do banco, exemplo, no Microsoft Excel como planilha:

```
COPY CRM_PROCESSO_XML  
TO 'C:/EMATEMP/TABELA.csv' CSV;
```

No exemplo acima, exportei todas as NFe importadas no DOX para um arquivo CSV.

## **IMPORTANTE:**

- *O arquivo é gerado no servidor onde esta rodando o serviço do postgresQL.*
- *O diretorio precisa existir.*
- *O diretorio precisa ter permissão para leitura e gravação.*

# Função - Formata CNPJ

SQL de criação:

```
create or replace function formata_cnpj (xcnpj in varchar) returns varchar(18) as
$body$
declare
    retorno varchar(18);
begin

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

    return retorno;
end;
$body$
language plpgsql;
```

Chamada da função:

```
select formata_cnpj('12345678000910')
```

Retorno:

```
12.345.678/0009-10
```



# Função - Formata CPF

SQL de criação:

```
create or replace function formata_cpf (xcpf in varchar) returns varchar(11) as
$body$
declare
    retorno varchar(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;

    return retorno;
end;
$body$
language plpgsql;
```

Chamada da função:

```
select formata_cpf('12345678910')
```

Retorno:

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

# Função - Primeiro e ultimo dia do mês

Olá, neste fórum iremos apresentar duas funções do PostgreSQL que podem ser utilizadas para retornar o primeiro e último dia do mês corrente.

## Primeiro dia do mês

```
SELECT cast(date_trunc(' month', current_date) as date);
```

## Último dia do mês

```
SELECT cast(date_trunc(' month', current_date) + INTERVAL'1 month' - INTERVAL'1 day' as date);
```

Nesta última query foi utilizado o comando **INTERVAL**, para saber mais sobre esta função, acesse a documentação oficial do PostgreSQL [clcando aqui](#).

Retorno da query executada:

[image-1646767758231.png](#)

Image not found or type unknown

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

SQL de criação da função:

```
create or replace function retorna_parte (xtexto in text, xdesde in text, xate in text, xcont
in int4) returns text as $body$
declare
    info text;
begin
    [select substr(split_part(xtexto, xdesde, xcont+1), 1, strpos(split_part(xtexto, xdesde,
xcont+1), xate)-1)
        into info;
    return info;
end;
$body$
language plpgsql;
```

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
```

Retorno:

	NOME		DATA		PROFISSAO	
--	------	--	------	--	-----------	--

-----  
| João da Silva | 30/03/1990 | Consultor D0X |

# Função de agregação (count, sum, max, min, avg, 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 INT4,  
  descricao TEXT,  
  tipo VARCHAR(20),  
  quantidade DECIMAL(15,2));  
  
INSERT INTO ex_itens VALUES (1, 'Biscoito', 'Comida', 20),  
                             (2, 'Água', 'Bebida', 47.95),  
                             (3, 'Suco', 'Bebida', 23.4),  
                             (4, 'Pão', 'Comida', 10),  
                             (5, 'Refrigerante', 'Bebida', 13.75);  
  
COMMIT;
```

## Consulta com agrupamento agregado:

```
SELECT SUM( QUANTIDADE)  VOLUME_POR_TIPO,
       MIN( QUANTIDADE)  MENOR_QTD_POR_TIPO,
       MAX( QUANTIDADE)  MAIOR_QTD_POR_TIPO,
       TRUNC( AVG( QUANTIDADE), 2)  MEDIA_QTD_POR_TIPO
FROM EX_ITENS
GROUP BY TIPO
```

Retorno:

QTD_POR_TIPO	VOLUME_POR_TIPO	MENOR_QTD_POR_TIPO	MAIOR_QTD_POR_TIPO	MEDIA_QTD_POR_TIPO
3	85.10	13.75	47.95	28.36
2	30	10	20	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;
- **Avg:** Retorna média 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,
       MIN( QUANTIDADE) OVER ( PARTITION BY TIPO) MENOR_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
FROM EX_ITENS
```

Retorno:

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

# Função de Strings

## Concatenação de strings - dois || (pipes)

```
SELECT 'ae' || 'io' || 'u' --> Retorno = 'aeiou'
```

```
SELECT CHR(67)|| CHR(65)|| CHR(84) --> Retorno = 'CAT'
```

## Quantidade de caracteres de string

```
SELECT CHAR_LENGTH(' UNIFORM') --> Retorno = 6
```

```
SELECT LENGTH(' Database') --> Retorno = 8
```

## Converter para minúsculas

```
SELECT LOWER(' UNIFORM') --> Retorno = 'uniform'
```

## Converter para maiúsculas

```
SELECT UPPER('universidade') --> Retorno = 'UNIVERSIDADE'
```

## Posição de caractere

```
SELECT POSITION ('@' IN 'bando@emasoftware.com.br') --> Retorno = 6
```

```
SELECT STRPOS('Ema Software' , 'Soft') --> Retorno = 5
```

## Substring

```
SELECT SUBSTRING ('Ema DOX' FROM 5 FOR 3) --> Retorno = 'DOX'
```

```
SELECT SUBSTRING ('PostgreSQL' FROM '.....') --> Retorno = 'Postgre'
```

```
SELECT SUBSTRING ('PostgreSQL' FROM '...$') --> Retorno = 'SQL' (últimos caracteres da String)
```

```
SELECT SUBSTR ('Ema Software' , 5, 8) --> Retorno = 'Software'
```



## Substituir todos os caracteres semelhantes

```
SELECT TRANSLATE('Brasil', 'il', 'ão') --> Retorno = 'Brasão'
```

```
SELECT TRANSLATE('Brasileiro', 'eiro', 'eira') --> Retorno = 'Brasileira'
```

## Remover espaços de strings

```
SELECT TRIM(' SQL - PADRÃO ') --> Retorno = 'SQL - PADRÃO' (remove espaços de ambos os lados da string)
```

```
SELECT RTRIM(' SQL - PADRÃO ') --> Retorno = ' SQL - PADRÃO' (remove espaços do lado direito da string)
```

```
SELECT LTRIM(' SQL - PADRÃO ') --> Retorno = 'SQL - PADRÃO ' (remove espaços do lado esquerdo da string)
```

## Calcular MD5 de String

```
SELECT MD5('Ema Software') --> Retorno = '50579faffec425086c8dc0f9fdbfd3be'
```

## Repetir uma string N vezes

```
SELECT REPEAT('SQL-', 3); - - Retorna SQL-SQL-SQL-
```

## Sobrescrever substring em string

```
SELECT REPLACE('Postgresql', 'sql', 'SQL') --> Retorno = 'PostgreSQL'
```

## Dividir cadeia de caracteres com delimitador

```
SELECT SPLIT_PART('Guarda-chuva', '-', 1) --> Retorno = 'Guarda'
```

```
SELECT SPLIT_PART('Guarda-chuva', '-', 2) --> Retorno = 'chuva'
```

## Iniciais maiúsculas

```
SELECT INITCAP('nome sobrenome') --> Retorno = 'Nome Sobrenome'
```

# Funções de data

## 1 - Funções básicas:

```
CURRENT_DATE - Data Atual  
CURRENT_TIME - Hora Atual  
CURRENT_TIMESTAMP - Data e Hora Atual
```

## 2 - Funções DATE\_TRUNC:

```
DATE_TRUNC - Especificar parte ser truncada em data informada
```

Com essa função é possível obter o parametro informado, ignorando os demais dados referentes a data, sua utilização se dá `date_trunc('CAMPO', ORIGEM)`, onde CAMPO é o tipo de dado e ORIGEM a data a qual deve ser truncada, abaixo lista de todos os CAMPOS disponíveis:

- microseconds
- milliseconds
- second
- minute
- hour
- day
- week
- month
- quarter
- year
- decade
- century
- millennium

## 3 - Exemplos de uso:

```
SELECT date_trunc(' hour', TIMESTAMP '2020-01-16 20:38:40');  
Result: 2020-01-16 20:00:00
```

```
SELECT date_trunc(' year', TIMESTAMP '2020-02-16 20:38:40');  
Result: 2020-01-01 00:00:00
```

Truncar para obter 1º dia do mês

```
date_trunc(' month', current_date)
```

# Gerar log dos comandos

Neste tópico veremos como configurar o **PostgreSQL** para gerar um log de todos os comando executados.

**\*\* Atenção :** Para realizar este procedimento os serviços do postgresQL devem estar parados, para isso basta acessar o gerenciador de tarefas e parar os serviços ou pressionar windows + R e digitar services.msc e procurar os serviços do postgres e parar eles.

Para que o **PostgreSQL** gere o log devemos configurar o arquivo **postgresql.conf**. O arquivo se encontra no seguinte caminho do servidor em que está instalado o banco de dados :

**C:\Program Files\PostgreSQL\10\data.**

image-1646765454295.png

Image not found or type unknown

Após acessar está pasta Edite o arquivo mencionado anteriormente, você pode utilizar o bloco de notas ou um editor como o Notepad++.

Ao editar procure pelos seguintes comando e preencha da seguinte forma:

```
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_statement = 'all'
logging_collector = on
```

**\*\*OBSERVAÇÃO:** Caso esses parâmetros estejam com # na frente do nome, isso significa que eles estão comentados e o postgresQL não irá executar eles, após descomentar basta salvar o arquivo.

Feito estes procedimentos basta iniciar novamente os serviços do PostgreSQL e na pasta Log ele irá gerar o arquivo atualizando ele com os comando executados no sistema. Ele irá gerar log até dos comando que são executadas em ferramenta de terceiro, qualquer interação ele irá logar.

# 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 VARCHAR(2), descricao TEXT);
INSERT INTO ex_estado VALUES ('SC', 'Santa Catarina'),
                              ('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 TEXT, uf VARCHAR(2));  
INSERT INTO ex_cidade VALUES (1, 'Criciúma', 'SC'),  
                                (2, 'Florianópolis', 'SC'),  
                                (3, 'Curitiba', 'PR');  
  
COMMIT;
```

# INNER JOIN

image-1646740796923.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 Catarina	Florianópolis

## LEFT OUTER JOIN

[image-1646744385807.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 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  
LEFT JOIN EX_CIDADE CI USING (UF)
```

Retorno:

ESTADO	CIDADE
Santa Catarina	Criciúma
Santa Catarina	Florianópolis
São Paulo	(null)

## RIGHT OUTER JOIN

image-1646744708047.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
Santa Catarina	Criciúma
(null)	Curitiba

## FULL OUTER JOIN

image-1646744999045.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 e Having

Com os operadores **ORDER BY**, **GROUP BY** e **HAVING** 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 INT4,  
  descricao TEXT,  
  tipo TEXT,  
  quantidade DECIMAL(15,2));  
INSERT INTO ex_itens VALUES (1, 'Biscoito', 'Comida', 20),  
                             (2, 'Água', 'Bebida', 47.95),  
                             (3, 'Suco', 'Bebida', 23.4),  
                             (4, 'Pão', 'Comida', 10),  
                             (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
2	' Água'	' Bebida'	47, 95
5	' Refrigerante'	' Bebida'	13, 75
1	' Biscoito' ☐	' Comida'	20
4	' Pão'	' Comida'	10

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

# Pacote de Funções DOX (Schema) - PostgreSQL

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 schema / package para criar no banco de dados que terá todas as funções e procedures.

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

- **Primeiramente, executar o script para criação do schema**

```
create schema pkg_ema authorization postgres; grant all on schema pkg_ema to postgres with grant option;
```

- **Em seguida, executar a criação das funções**

```
CREATE OR REPLACE FUNCTION pkg_ema.retorna_lista(in_lista text, delimitador character)
  RETURNS TABLE(out_id integer, out_str text)
AS $body$
declare
  dados record;
begin
  IN_LISTA = REPLACE(REPLACE(IN_LISTA, '(', ''), ')', '');

  IF DELIMITADOR = '' OR DELIMITADOR IS NULL THEN DELIMITADOR = ','; END IF;
  IF IN_LISTA <> '.' and IN_LISTA is not null and IN_LISTA <> '' and (IN_LISTA not like '/*%'
and IN_LISTA not like '%*/') THEN
    for dados in (select unnest(string_to_array(IN_LISTA, DELIMITADOR)) LISTA)
    LOOP
      BEGIN
        OUT_ID  = cast(dados.lista as integer);
```

```

        OUT_STR = dados.lista;
RETURN NEXT;

EXCEPTION
WHEN OTHERS then
    OUT_ID = 0;
OUT_STR = dados.lista;
    RETURN NEXT;
END;
END LOOP;
ELSE
OUT_ID = 0;
    OUT_STR = '0';
    RETURN NEXT;
END IF;

end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_lista(text, character) owner to postgres;

create or replace function pkg_ema.retorna_valor_variavel(xidprocesso in int4, xidvariavel in
int4) returns varchar as
$body$
    declare
    valor varchar(5000);
    begin
    select valoratual into valor
    from public.crm_processo_variavel
    where.crm_processo_variavel.idprocesso = xidprocesso
    and.crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
    end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel(int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_valor_variavel_texto(xidprocesso in int4,
xidvariavel in int4) returns varchar as
$body$

```



```

declare
valor varchar(5000);
begin
select valoratual into valor
from public.crm_processo_variavel
where crm_processo_variavel.idprocesso = xidprocesso
and crm_processo_variavel.idvariavel = xidvariavel ;
return valor;
end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_texto(int4, int4) owner to postgres;

```

```

CREATE OR REPLACE FUNCTION pkg_ema.retorna_valor_variavel_clob(xidprocesso integer,
xidvariavel integer) RETURNS text as

```

```

$body$
declare
valor text;
begin
select valoratual into valor
from public.crm_processo_variavel
where crm_processo_variavel.idprocesso = xidprocesso
and crm_processo_variavel.idvariavel = xidvariavel ;
return trim(valor);
end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_clob(int4, int4) owner to postgres;

```

```

create or replace function pkg_ema.retorna_valor_variavel_int(xidprocesso in int4,
xidvariavel in int4) returns int4 as

```

```

$body$
declare
valor int4;
begin
select cast(case when valoratual = '.' then '0' else valoratual end as int4)
into valor
from public.crm_processo_variavel
where crm_processo_variavel.idprocesso = xidprocesso

```

```

    and crm_processo_variavel.idvariavel = xidvariavel ;
    return valor;
end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_int(int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_valor_variavel_valor(xidprocesso in int4,
xidvariavel in int4) returns decimal(15,2) as
$body$
    declare
        valor decimal(15,2) ;
    begin
        select replace(replace(case when valoratual = '.' then '0' else valoratual end, '.', ''),
',', '.')
        into valor
        from public.crm_processo_variavel
        where crm_processo_variavel.idprocesso = xidprocesso
        and crm_processo_variavel.idvariavel = xidvariavel ;
        return valor;
    end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_valor(int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_valor_variavel_data(xidprocesso in int4,
xidvariavel in int4) returns date as
$body$
    declare
        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 public.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;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_data(int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_valor_variavel_datahr(xidprocesso in int4,
xidvariavel in int4) returns timestamp as
$body$
    declare
        valor timestamp;
    begin
        if (xidprocesso > 0 and xidvariavel > 0) then
            begin
                select case when valoratual <> '.' then to_timestamp(valoratual,'dd/mm/yyyy HH24:MI')
                    else to_timestamp('01-01-1900','dd/mm/yyyy HH24:MI') end into valor
                from public.crm_processo_variavel
                where crm_processo_variavel.idprocesso = xidprocesso
                and crm_processo_variavel.idvariavel = xidvariavel ;
            end;
        else
            valor := to_timestamp('01/01/1900','dd/mm/yyyy HH24:MI');
        end if;
        return valor;
    end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_valor_variavel_datahr(int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_coluna_grade(xidprocesso int4, xidatividade int4,
xidformulario int4, xidgrade int4, xidvalor int4)
returns character varying as
$body$
    declare
        leitura varchar(4000);
    begin
        select
            resposta into leitura

```

```

from public.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(trim(leitura));
end;
$body$

language plpgsql volatile cost 100;
alter function pkg_ema.retorna_coluna_grade(int4, int4, int4, int4, int4) owner to postgres;

create or replace function pkg_ema.retorna_coluna_grade_texto(xidprocesso int4, xidatividade
int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns character varying as
$body$
declare
leitura varchar(4000);
begin
select
resposta into leitura
from public.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(trim(leitura));
end;
$body$

```

```

language plpgsql volatile cost 100;

alter function pkg_ema.retorna_coluna_grade_texto(int4, int4, int4, int4, int4) owner to
postgres;

create or replace function pkg_ema.retorna_coluna_grade_clob(xidprocesso int4, xidatividade
int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns text as
$body$
declare
    leitura text;
begin
    select
        resposta into leitura
    from public.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(trim(leitura));
end;
$body$

language plpgsql volatile cost 100;

alter function pkg_ema.retorna_coluna_grade_texto(int4, int4, int4, int4, int4) owner to
postgres;

create or replace function pkg_ema.retorna_coluna_grade_int(xidprocesso int4, xidatividade
int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns int4 as
$body$
declare
    leitura int4;

begin
    /*leitura := 0 ;

```

```

select (select to_number(decode(respostainteiro,'.',0,respostainteiro)) from dual)*/

select respostainteiro into leitura
from public.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);

return(leitura);

end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_coluna_grade_int(int4, int4, int4, int4, int4) owner to
postgres;

create or replace function pkg_ema.retorna_coluna_grade_valor(xidprocesso int4, xidatividade
int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns numeric(15,2) as
$body$
declare
    leitura numeric(15,2);
begin
    select respostavalor into leitura
    from public.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);

```

```

    return(leitura);
end;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_coluna_grade_valor(int4, int4, int4, int4, int4) owner to
postgres;

create or replace function pkg_ema.retorna_coluna_grade_data( xidprocesso int4, xidatividade
int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns date as
$body$
declare
    leitura date;
begin
    select respostadatahora into leitura
    from public.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); --fazer o trim depois

end;
$body$

language plpgsql volatile cost 100;
alter function pkg_ema.retorna_coluna_grade_data(int4, int4, int4, int4, int4) owner to
postgres;

create or replace function pkg_ema.retorna_coluna_grade_datahr( xidprocesso int4,
xidatividade int4, xidformulario int4, xidgrade int4, xidvalor int4)
returns timestamp as
$body$
declare

```

```

    leitura timestamp;
begin
    select respostadatahora into leitura
    from public.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;
$body$
language plpgsql volatile cost 100;
alter function pkg_ema.retorna_coluna_grade_datahr(int4, int4, int4, int4, int4) owner to
postgres;

```

## Chamando as Funções

- 

### Retorna lista

	RETORNO	
SQL	OUT_ID (Saída Integer)	OUT_STR (Saída Text)
select * from pkg_ema.retorna_lista('/*IDCLIENTE*/','');	0	0
select * from pkg_ema.retorna_lista('0','');	0	0
select * from pkg_ema.retorna_lista('','');	0	0
select * from pkg_ema.retorna_lista('(9,7)','','');	9 7	9 7
select * from pkg_ema.retorna_lista('(10;74)','','');	10 74	10 74



	RETORNO	
select * from pkg_ema.retorna_lista('(91:5:S:SS)',':'); 0	91 5 0 0	91 5 S SS
select * from pkg_ema.retorna_lista('(ASD,ASDA,ASDA)',','); 0	0 0 0	ASD ASDA ASDA

- **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,
       pkg_ema.retorna_valor_variavel_clob(idprocesso, 7) as clob
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,
       pkg_ema.retorna_coluna_grade_clob(x.idprocesso, x.idatividade, x.idformulario, 7,
x.idvalor) as clob
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
```



```
WHERE IDPAPELFUNCAO = 25
```

```
UNION ALL
```

```
SELECT IDUSUARIO
```

```
FROM USUARIO
```

```
WHERE USERNAME = ' /*USUARIO*/ ', ' ' )
```

```
FROM VERSA0DB
```

Perceba que neste caso, optei por fazer um union all dentro do array, assim o resultado será convertido e concatenado ao conjunto de resultado, existem outras formas de atender essa demanda, espero que tenha ajudado e que você possa adaptar os exemplos acima em seus processos / SQLs.

# Retorna lista

Código SQL a ser usado:

```
select unnest(string_to_array('1,2,3', ',')) as lista
```

Retorno:

LISTA
1
2
3

# Tamanhos da tabelas (MB)

Neste tópico mostraremos o comando **SQL** que retorna qual **schema**, **tabela**, e tamanho em **MB** de cada uma delas.

```
SELECT esquema, tabela,
       pg_size_pretty(pg_relation_size(esq_tab)) AS tamanho,
       pg_size_pretty(pg_total_relation_size(esq_tab)) AS tamanho_total
FROM (SELECT tablename AS tabela,
             schemaname AS esquema,
             schemaname||'.'||tablename AS esq_tab
      FROM pg_catalog.pg_tables
      WHERE schemaname NOT
            IN ('pg_catalog', 'information_schema', 'pg_toast') ) AS x
ORDER BY pg_total_relation_size(esq_tab) DESC;
```

Retorno do banco:

[image-1646765251845.png](#)

Image not found or type unknown

# Union, Union All, Except, Except All e Intersect

É possível combinar os resultados de duas ou mais consultas através dos operadores **Union**, **Except** 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'
'Matriz'
'Filial 1'
'Filial 1'
'Filial 2'
'Filial 3'

### SQL de CRIAÇÃO/INSERÇÃO:

```
CREATE TABLE ex_filial (descricao TEXT);
INSERT INTO ex_filial VALUES ('Matriz'),
                              ('Matriz'),
                              ('Filial 1'),
                              ('Filial 1'),
                              ('Filial 2'),
                              ('Filial 3');

COMMIT;
```

### EX\_PONTOS\_DE\_VENDA

DESCRICAO
'Filial 1'

```
| 'Filial 2' |  
| 'Filial 2' |  
| 'Filial 3' |  
| 'Unidade 1' |  
| 'Unidade 1' |  
| 'Unidade 2' |
```

SQL de CRIAÇÃO/INSERÇÃO:

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

## UNION

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

```
SELECT DESCRICAO FROM EX_FILIAL  
UNION  
SELECT DESCRICAO FROM EX_PONTOS_DE_VENDA
```

Retorno:

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

## UNION ALL

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



```
SELECT DESCRICAO FROM EX_FILIAL
UNION ALL
SELECT DESCRICAO FROM EX_PONTOS_DE_VENDA
```

Retorno:

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

## EXCEPT

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

```
SELECT DESCRICAO FROM EX_FILIAL
EXCEPT
SELECT DESCRICAO FROM EX_PONTOS_DE_VENDA
```

Retorno:

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

## EXCEPT ALL

O operador **EXCEPT ALL** elimina os dados da primeira consulta de acordo com a quantidade que o mesmo resultado se repete na segunda.

```
SELECT DESCRICAO FROM EX_FILIAL
```

```
EXCEPT ALL
SELECT DESCRICAO FROM EX_PONTOS_DE_VENDA
```

Retorno:

```
| DESCRICAO |
-----
| 'Filial 1' |
| 'Matriz'   |
| '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, timestamp, to\_char, etc)

## CURRENT\_DATE

Selecionada data atual

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

## CURRENT\_TIME

Seleciona hora atual

```
select current_time --> Retorno: "08:56:29.649098-03"
```

## CURRENT\_TIMESTAMP

Seleciona data e hora atual (funções alternativas: now() e transaction\_timestamp())

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

**OBS:** 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') --> Retorno: "09/2019"
```

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

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

```
select to_char(current_date, 'TMDay') --> Retorno: "Sexta-Feira"
```

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

## 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') --> Retorno: "2019-09-13"
```

```
select to_timestamp('13/09/2019 08:51:43', 'DD/MM/YYYY HH24:MI:SS') --> 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(millennium from now()) --> Retorno: 3 <-- (Milênio atual)
```

```
select extract(century from now()) --> Retorno: 21 <-- (século atual)
```

```
select extract(decade from now()) --> Retorno: 201 <-- (década atual)
```

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

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

```
select extract(dow from now()) --> Retorno: 5 <-- (dia da semana atual)
```

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

```
select extract(hour from now())||':'||
```

```
extract(minute from now())||':'||
```

```
trunc(extract(second from now())) --> Retorno: 14:43:46 <-- (hora atual)
```

## CAST

Função para fazer conversão de valores de diversos tipos. Neste caso, de data para texto de texto

para data.

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

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

# With, Substr, Split\_part e Strpos

Neste exemplo será mostrado como utilizar as funções **With**, **Substr** e **Split\_part** e **Strpos** 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'
```

## SQL:

```
WITH nota AS (SELECT 'NF 123456 - DT 07/10/2019 - VL 37,99' AS dados)
SELECT substr(split_part(dados, 'NF', 2), 2, strpos(split_part(dados, 'NF', 2), '-')-3) AS
numeronf,
       substr(split_part(dados, 'DT', 2), 2, strpos(split_part(dados, 'DT', 2), '-')-3) AS
dataemissao,
       substr(split_part(dados, 'VL', 2), 2) 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;
- **Split\_part:** retorna parte do texto a partir do caractere ou conjunto de caracteres informados;
- **Strpos:** retorna a posição do texto, onde se encontra o caractere ou conjunto de caracteres informado, para usar a posição no **Substr**;