



Você está em

Guia de PostgreSQL » Views

Artigo

Como funcionam as Views no PostgreSQL

Veja nesse artigo como utilizar Views simples e Views materializadas no banco de dados PostgreSQL 9.4.



Marcado como lido



Anotar

Artigos



Banco de Dados



Como funcionam as Views no PostgreSQL

Neste artigo trataremos da criação e exclusão das Views no **PostgreSQL**.

Conheceremos sua estrutura básica e aprenderemos a usá-las em **nossas consultas** para obter resultados de forma simplificada.

Primeiramente, precisamos entender o que são as Views, que são consideradas





Você está em

Guia de PostgreSQL » Views

Vamos começar apresentando a sintaxe básica de uma view, conforme a **Listagem 1**.

Listagem 1. Sintaxe de criação de uma View.

```
1 CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW name
2   [ ( column_name [, ...] ) ]
3   [ WITH ( view_option_name [= view_option_value] [, ...] ) ]
4   AS query
5   [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

O código começa com a instrução de criação CREATE VIEW. E como ela é virtual, sempre que uma consulta é executada, a View é referenciada na consulta.

A instrução CREATE [OR REPLACE] VIEW é semelhante a anterior, mas verifica se já existe uma view com o mesmo nome e, caso exista, será substituída pela mais recente. O nome dado a nova view não pode ser o mesmo já atribuído também a qualquer outro item presente no banco.

Caso o nome do Schema tenha sido fornecido, a View será criada com base nele e não no Schema global. Quando temos um Schema sendo especificado, este não pode ser temporário, assim como as Views.

O parâmetro [TEMPORARY | TEMP] é especificado quando queremos que a View seja temporária, as quais são eliminadas de forma automática quando a sessão atual é finalizada. Caso tenhamos referenciado tabelas temporárias para a View, esta será criada como temporária mesmo que não tenhamos especificado este parâmetro.





Você está em

Guia de PostgreSQL » Views

Podemos também utilizar a seguinte sintaxe, que é uma versão reduzida:

```
1 | CREATE RECURSIVE VIEW name (columns) AS SELECT ...;
```

Um exemplo real da aplicação desse conceito seria ter uma tabela de usuários, onde procuraríamos dentre estes quais são administradores:

```
1 | CREATE VIEW administradores AS WITH RECURSIVE usuarios (columns) AS (S
```

O Name é o nome da View, podendo este ser opcional já o column_name, que é uma lista opcional que contém os nomes a serem utilizados para as colunas da View.

O check_option (string), pode ser local ou em cascata, e é equivalente à quando especificamos o WITH [CASCADED | LOCAL] CHECK OPTION. Esta opção pode ser alterada em Views existentes quando utilizamos a instrução ALTER VIEW. O security_barrier (string) é utilizado quando a View é criada com o intuito de fornecer segurança a nível de linha.

O WITH [CASCADE | LOCAL] CHECK OPTION é uma opção utilizada para controle do comportamento da View com base nas atualizações automáticas. No caso desta opção ser especificada, os comandos de Insert e Update são verificados para garantir que as novas linhas satisfaçam as condições que definem a View. Caso não estejam em conformidade, a atualização não ocorrerá. Caso o CHECK OPTION não seja





Você está em

Guia de PostgreSQL » Views

- LOCAL – onde novas linhas são verificadas apenas contra as condições definidas diretamente na própria View.
- CASCADED – aqui novas linhas são verificadas em relação às condições da View e todas as Views subjacentes.

Se a cláusula CHECK OPTION for especificada e não tivermos a especificação da LOCAL nem da CASCADED, então o CASCADED é assumido. Precisamos ter cuidado apenas com as Views recursivas, pois o CHECK OPTION não pode ser utilizado nelas. Ou seja, esse parâmetro só é suportado apenas em Views que são atualizadas automaticamente e não possuem triggers do tipo Instead Of ou mesmo que contenham regras do tipo Instead. Se uma View que é atualizada automaticamente for definida com base em uma View que contenha uma trigger Instead of, então a LOCAL CHECK OPTION pode ser utilizada para verificar as condições sobre esta View, mas as condições referentes a View que contém a trigger INSTEAD OF não será verificada.

Vamos criar uma tabela funcionarios que será responsável por manter os registros reais, como mostra a **Listagem 2**.

Listagem 2. Criando a tabela de funcionários.

```
1 CREATE TABLE funcionarios
2 (
3     codigo integer NOT NULL,
4     nome_func character varying(100) NOT NULL,
```





Você está em

Guia de PostgreSQL » Views

```
13 | ALTER TABLE funcionarios  
14 | OWNER TO postgres;
```

Vamos criar uma view para essa tabela usando a sintaxe a seguir:

```
1 | CREATE VIEW view_funcionarios AS SELECT * FROM Funcionarios;
```

A view_funcionarios contém uma instrução SELECT para receber os dados da tabela. Para que possamos ver as views que criamos, podemos utilizar a seguinte declaração:

```
1 | SELECT codigo, nome_func, profissao FROM view_funcionarios;
```

Uma View é um objeto que permite a visualização de dados da tabela a qual esteja associada. Como ela não existe por conta própria, é criada com base em consultas nas tabelas para selecionar colunas, dando assim acessos restritos ou com privilégios. Além disso possuem a capacidade de juntar informações contidas em diversas tabelas para representar em um único lugar como, por exemplo, na geração de relatórios.

Devido as suas especificações de restrição, os dados retornados são apenas aqueles que o usuário pode ver. E por serem tabelas virtuais, não podemos realizar por elas as operações DML de inserção, atualização e exclusão, mas sim apenas usando o SELECT.





Você está em

Guia de PostgreSQL » Views

Listagem 3. Criação da tabela de registro_ponto.

```
1 CREATE TABLE registro_ponto
2 (
3     registro_ponto_id integer NOT NULL,
4     hora_entrada time without time zone,
5     "codFunc" integer NOT NULL,
6     entrada date,
7     CONSTRAINT registro_ponto_pkey PRIMARY KEY (registro_ponto_id),
8     CONSTRAINT "codFuncFK" FOREIGN KEY ("codFunc")
9         REFERENCES funcionarios (codigo) MATCH SIMPLE
10        ON UPDATE NO ACTION ON DELETE NO ACTION
11 )
12 WITH (
13     OIDS=FALSE
14 );
15 ALTER TABLE registro_ponto
16     OWNER TO postgres;
17
18 -- Index: "fki_codFuncFK"
19
20 -- DROP INDEX "fki_codFuncFK";
21
22 CREATE INDEX "fki_codFuncFK"
23     ON registro_ponto
24     USING btree
25     ("codFunc");
```

Agora criaremos uma segunda view View_Ponto_funcionario que utilizará o código do funcionário na cláusula where, como vemos a seguir:





Você está em

Guia de PostgreSQL » Views

```
1 | SELECT * FROM View_Ponto_funcionario;
```

Para vermos o resultado inseriremos alguns registros em ambas as tabelas, como mostra a **Listagem 4**.

Listagem 4. Inserção de dados nas tabelas funcionarios e registro_ponto.

```
1 | INSERT INTO funcionarios(codigo, nome_func, data_entrada, profissao, s
2 |     INSERT INTO funcionarios(codigo, nome_func, data_entrada, profissao)
3 |     INSERT INTO funcionarios(codigo, nome_func, data_entrada, profissao)
4 |     INSERT INTO registro_ponto(registro_ponto_id, entrada, hora_entrada,
5 |     INSERT INTO registro_ponto(registro_ponto_id, entrada, hora_entrada,
6 |     INSERT INTO registro_ponto(registro_ponto_id, entrada, hora_entrada,
7 |     Feita a inserção dos dados, podemos ver as mudanças na nossa segunda
8 |     SELECT * FROM View_Ponto_funcionario;
```

Como mostra a **Figura 1**, as informações sendo apresentadas na View contendo os dados solicitados das duas tabelas.

Data Output	Explain	Messages	History		
	nome_func character varying(100)	profissao character varying(100)	entrada date	hora_entrada time without time zone	
1	Edson Dionisio	Desenvolvedor Web	2015-11-09	08:00:00	
2	Edson Dionisio	Desenvolvedor Web	2015-10-01	13:00:00	
3	Marilia Késsia	Coordenadora	2015-10-06	08:00:00	

Figura 1. Resultado da consulta a View View_Ponto_funcionario.

Para excluir uma view usamos o seguinte comando:





Você está em

Guia de PostgreSQL » Views

armazenados nas tabelas do banco de dados, podemos “materializadas”, ou seja, armazená-las fisicamente no disco.

A criação dessas Views, ao invés de novas tabelas, melhora o desempenho em operações de leitura. Os dados das Views materializadas serão então armazenados em uma tabela que pode ser indexada rapidamente quando esta for associada e também em momentos que precisemos atualizar as Views materializadas. Isso ocorre com frequência em Data warehouse e aplicações de Business Intelligence, por exemplo.

Ao contrário da View tradicional, que nos apresentam dados atualizados automaticamente, as Views materializadas precisam de um mecanismo de atualização. A sintaxe é apresentada na **Listagem 5**.

Listagem 5. Sintaxe básica de uma View materializada.

```
1 CREATE MATERIALIZED VIEW table_name
2     [ (column_name [, ...] ) ]
3     [ WITH ( storage_parameter [= value] [, ...] ) ]
4     [ TABLESPACE tablespace_name ]
5     AS query
6     [ WITH [ NO ] DATA ]
```

A instrução CREATE MATERIALIZED VIEW cria View e a consulta é executada para preenche-la. Para atualizar os dados usamos o comando REFRESH MATERIALIZED VIEW. Uma View materializada tem muitas das mesmas propriedades de uma





Você está em

Guia de PostgreSQL » Views

será realizada no default_tablespace.

Com base no exemplo que desenvolvemos nesse artigo, a instrução a seguir cria uma view materializada para a tabela funcionarios:

```
1 | CREATE MATERIALIZED VIEW view_materializada_funcionario AS SELECT * FROM
```

Para inserir dados nessa view usaremos o código da **Listagem 6**.

Listagem 6. Inserindo dados na view materializada

```
1 | INSERT INTO funcionarios (codigo, nome_func, data_entrada, profissao) '  
2 |     INSERT INTO funcionarios (codigo, nome_func, data_entrada, profissao  
3 |     INSERT INTO funcionarios (codigo, nome_func, data_entrada, profissao  
4 |     INSERT INTO funcionarios (codigo, nome_func, data_entrada, profissao  
5 |     INSERT INTO funcionarios (codigo, nome_func, data_entrada, profissao
```

Agora utilizaremos o comando SELECT para verificarmos se os registros foram realmente inseridos com sucesso:

```
1 | SELECT * FROM view_materializada_funcionario;
```

Podemos ver que obtivemos um erro ao tentarmos consultar a nossa View, como mostra a **Figura 2**.





Você está em

Guia de PostgreSQL » Views

Figura 2. Visualização de erro ao consultar View materializada.

Este erro ocorreu porque a view não se atualizou automaticamente, então precisamos do comando Refresh Materialized View:

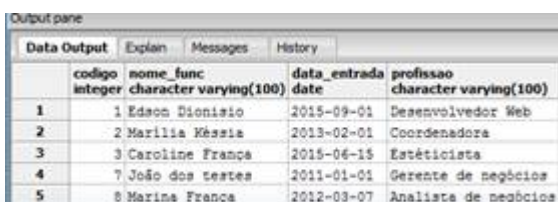
```
1 | REFRESH MATERIALIZED VIEW view_materializada_funcionario;
```

Feito isso, e em seguida, tentando consultar novamente os dados e assim obtemos êxito.

Com o PostgreSQL 9.4 podemos consultar Views materializadas enquanto são atualizadas, porém, nas versões anteriores isso não é possível. Para esse procedimento utilizamos a palavra-chave CONCURRENTLY, como mostra o comando a seguir:

```
1 | REFRESH MATERIALIZED VIEW CONCURRENTLY view_materializada_funcionario;
```

Para que isso ocorra, um índice exclusivo passa a ser necessário para existir na View materializada. Sendo assim, ao executarmos o comando select novamente, teremos todos os dados atualizados, como podemos ver na **Figura 3**.



	codigo integer	nome_func character varying(100)	data_entrada date	profissao character varying(100)
1	1	Edson Dionísio	2015-09-01	Desenvolvedor Web
2	2	Marília Kássia	2013-02-01	Coordenadora
3	3	Caroline França	2015-06-15	Esteticista
4	7	João dos testes	2011-01-01	Gerente de negócios
5	8	Marina França	2012-03-07	Analista de negócios





Você está em

Guia de PostgreSQL » Views

Todos os recursos de otimização são importantes, pois lembre-se que as views são apenas visões dos dados e são carregadas apenas quando necessárias. Mas elas criam uma camada extra para administrar e podem limitar exageradamente, impedindo certas tarefas.

Além disso, não confunda uma view materializada com uma trigger, pois apesar de funcionarem de forma semelhante, a trigger tem muito mais poder sobre a tabela.

Esperamos que tenham gostado. Até a próxima!

Links

Documentação da View

<http://www.postgresql.org/docs/9.4/static/sql-createview.html>



Marcado como lido



Anotar

Por **Edson**

Em 2015



9





Você está em

Guia de PostgreSQL » Views

Tecnologias

Exercícios

Cursos

Artigos

Revistas

Fale conosco

Trabalhe conosco

Assinatura para empresas

Assine agora



Hospedagem web por Porta 80 Web Hosting

