Views

Professor

Fernando Martins Medeiros

Views

VIEWS

são consideradas pseudo-tables, ou seja, elas são usadas junto a instrução SELECT para apresentar subconjuntos de dados presentes em tabelas reais. Assim, podemos apresentar as colunas e linhas que foram selecionadas da tabela original ou associada. E como as Views possuem permissões separadas, podemos utilizá-las para restringir mais o acesso aos dados pelos usuários, para que veja apenas o que é necessário.

Views

Vantagens na utilização de VIEWS

• Estas **consultas** pré-definidas ficam armazenadas e você não precisa lembrar de como criá-las.

• Como o próprio nome ajuda identificar elas permitem criar uma visão mais lógica para um humano entender a modelagem.

• Facilita a troca do modelo físico sem o perigo de quebrar *queries* existentes.

• Eventualmente o banco de dados **pode** fazer algumas otimizações por já ter conhecimento das *queries* utilizadas nas *views*.

• Você pode colocar permissões na *view*, ou seja, você pode proibir acesso à tabelas em seu estado bruto, mas em uma certa condição o usuário pode ter acesso à informação da forma como você definiu na *view*. Você controla melhor o que e como o usuário pode acessar a informação.

• Se a *view* for materializada pode ter ganho de performance para o acesso aos dados já "consolidados".

Views

Desvantagens na utilização de VIEWS

• Esconde uma complexidade da *query* podendo enganar o desenvolvedor quanto à performance necessária para acessar determinada informação. E pode ser pior quando *views* usam outras *views*. Em alguns casos você pode estar fazendo consultas desnecessárias sem saber de forma muito intensiva.

• Cria uma camada extra. Mais objetos para administrar. Algumas pessoas consideram isto um aumento de complexidade.

• Se a *view* for materializada fará com que alterações nas tabelas reais envolvidas não estejam disponíveis imediatamente no resultado.

Views no PostgreSQL

VIEWS

Sintaxe (Criação):

```
CREATE OR REPLACE VIEW ficha_funcionario AS
 SELECT
    funcionario.id,
    funcionario.nome,
    funcionario.cpf,
    funcionario.sexo,
    funcionario.data_nascimento,
    funcionario.email,
    funcionario.celular,
    funcionario.data_admissao,
    funcionario.salario
    FROM funcionario;
```

Execução:

```
SELECT *
FROM ficha_funcionario;

SELECT *
FROM ficha_funcionario
WHERE nome ILIKE '%martins%';
```

Views no PostgreSQL

VIEWS

	id integer	nome character varying(100)	cpf character varying(14)	sexo character(1)	data_nascimento date	email character varying(100)	celular character varying(30)	data_admissao date	salario numeric(10,2)
1	1	FERNANDO MARTINS	98928614007	M	1983-09-01	fernando.medeiros@edu.sc.senai.br	(48) 99545-5478	2000-08-01	5000.00
2	2	FELIPE CORREA	53972072090	M	1983-08-25	felipe.correa@edu.sc.senai.br	(48) 99545-5425	2017-08-01	2500.00
3	3	JOAO PAULO SILVA	51479258059	M	1982-05-02	joao.paulo@edu.sc.senai.br	(48) 99545-1237	2020-08-01	7000.00
4	4	MARIA MARTA SANTOS	22871185000	F	1988-03-20	maria.marta@edu.sc.senai.br	(48) 99545-9874	2021-08-01	3000.00
5	5	MARLENE MATOS	11821499018	F	1990-01-18	marlene.matos@edu.sc.senai.br	(48) 98545-5474	1999-08-01	9000.00

Views no PostgreSQL

VIEWS

Sintaxe (Criação):

```
CREATE OR REPLACE VIEW folha_ponto_funcionario AS
 SELECT
    funcionario.id,
    funcionario.nome,
    funcionario.data_admissao,
    registro_ponto.data_entrada,
    registro_ponto.hora_entrada,
    registro_ponto.data_saida,
    registro_ponto.hora_saida
FROM funcionario
INNER JOIN registro_ponto ON (funcionario.id = registro_ponto.id_funcionario)
ORDER BY funcionario.id, registro_ponto.data_entrada;
```

Execução:

```
SELECT *
FROM folha_ponto_funcionario;
```

```
SELECT *
FROM folha_ponto_funcionario
WHERE nome ILIKE '%martins%';
```

Views no PostgreSQL

VIEWS

	id integer	nome character varying(100)	data_admissao date	data_entrada date	hora_entrada time without time zone	data_saida date	hora_saida time without time zone
1	1	FERNANDO MARTINS	2000-08-01	2021-10-01	08:00:00	2021-10-01	18:00:00
2	1	FERNANDO MARTINS	2000-08-01	2021-10-04	08:00:00	2021-10-04	18:00:00
3	1	FERNANDO MARTINS	2000-08-01	2021-10-05	08:00:00	2021-10-05	18:00:00
4	1	FERNANDO MARTINS	2000-08-01	2021-10-06	08:00:00	2021-10-06	18:00:00
5	1	FERNANDO MARTINS	2000-08-01	2021-10-07	08:00:00	2021-10-07	18:00:00
6	1	FERNANDO MARTINS	2000-08-01	2021-10-08	08:00:00	2021-10-08	18:00:00
7	1	FERNANDO MARTINS	2000-08-01	2021-10-11	08:00:00	2021-10-11	18:00:00
8	1	FERNANDO MARTINS	2000-08-01	2021-10-13	08:00:00	2021-10-13	18:00:00
9	1	FERNANDO MARTINS	2000-08-01	2021-10-14	08:00:00	2021-10-14	18:00:00
10	1	FERNANDO MARTINS	2000-08-01	2021-10-15	08:00:00	2021-10-15	18:00:00
11	1	FERNANDO MARTINS	2000-08-01	2021-10-18	08:00:00	2021-10-18	18:00:00
12	1	FERNANDO MARTINS	2000-08-01	2021-10-19	08:00:00	2021-10-19	18:00:00
13	1	FERNANDO MARTINS	2000-08-01	2021-10-20	08:00:00	2021-10-20	18:00:00
14	1	FERNANDO MARTINS	2000-08-01	2021-10-21	08:00:00	2021-10-21	18:00:00
15	1	FERNANDO MARTINS	2000-08-01	2021-10-22	08:00:00	2021-10-22	18:00:00
16	1	FERNANDO MARTINS	2000-08-01	2021-10-25	08:00:00	2021-10-25	18:00:00
17	1	FERNANDO MARTINS	2000-08-01	2021-10-26	08:00:00	2021-10-26	18:00:00
18	1	FERNANDO MARTINS	2000-08-01	2021-10-27	08:00:00	2021-10-27	18:00:00
19	1	FERNANDO MARTINS	2000-08-01	2021-10-28	08:00:00	2021-10-28	18:00:00
20	1	FERNANDO MARTINS	2000-08-01	2021-10-29	08:00:00	2021-10-29	18:00:00

VIEWS MATERIALIZADAS

Visões materializadas são recursos introduzidos na versão 9.3 do postgresql. Enquanto visões tradicionais reexecutam uma consulta sempre que são referenciadas, visões materializadas dispensam este esforço pelos seus dados já estarem guardados desde a sua criação ou do último refresh (atualização de visão). Pode-se dizer que uma visão materializada é um objeto que contém o resultado de uma consulta, facilitando o acesso aos dados nela contidos.

Views no PostgreSQL

VIEWS MATERIALIZADAS

Sintaxe (Criação):

```
CREATE MATERIALIZED VIEW vm_folha_ponto_funcionario AS
SELECT
    funcionario.id,
    funcionario.nome,
    funcionario.data_admissao,
    registro_ponto.data_entrada,
    registro_ponto.hora_entrada,
    registro_ponto.data_saida,
    registro_ponto.hora_saida
FROM funcionario
INNER JOIN registro_ponto ON (funcionario.id = registro_ponto.id_funcionario)
ORDER BY funcionario.id, registro_ponto.data_entrada
WITH NO DATA;
```

Atualização:

```
REFRESH MATERIALIZED VIEW vm_folha_ponto_funcionario;
```

Execução:

```
SELECT *
FROM vm_folha_ponto_funcionario;
```

```
SELECT *
FROM vm_folha_ponto_funcionario
WHERE nome ILIKE '%martins%';
```