PostgresFDW

Reginaldo Gregório de Souza Neto, Isaac Santana de Almeida

Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão, PR – Brazil

reginaldoneto@alunos.utfpr.edu.br, isaacalmeida@alunos.utfpr.edu.br

Resumo. *Tendo em vista a necessidade de utilização de diferentes bancos de dados em diferentes tipos de servidores e SGBDs, foi criado o FDW(Foreign Data Wrapper), que possibilita a visualização e integração de dados remotamente, ou seja, permite utilizar dados de determinado servidor,em outro servidor que contém tabelas de dados, sendo isso, em tipos diferentes de banco de dados. Além disso, o FDW possibilita a utilização de DML(linguagem de manipulação de dados), o que nos permite fazer a manipulação dos dados em diferentes SGBDs. O objetivo deste artigo, é apresentar um conteúdo didático sobre o funcionamento do Postgres FDW, os mecanismos de utilização e as extensões para diversos SGBDs , sejam eles banco de dados relacionais ou não.*

1. O que é o PostgreSQL?

O PostgreSQL é um Sistema de Gerenciamento de Banco de Dados (SGBD) relacional, open source desenvolvido em na universidade de Berkeley, Califórnia. Sendo publicado em sua primeira versão por dois estudantes, possuía o nome de Postgres 95. Atualmente, após diversas atualizações e contribuições da comunidade, se encontra versão 15 (beta), sendo muito semelhante ao mySQL que foi exposto na disciplina de Banco de Dados 1.

O PostgreSQL possui sintaxe e comandos muito parecidos a outros SQL´s, pois é um dos pioneiros em muitos conceitos que vieram a ser aplicados em outros SGBD´s atuais. Também é muito utilizado no ambiente corporativo e de desenvolvimento, devido a sua versatilidade por se tratar de um software livre, possibilita diversas melhorias e contribuições da comunidade, incluindo de grandes empresas como a Apple e a RedHat, que fomentaram upgrades no software, entretanto nenhuma delas é a dona do postgreSQL.

2. O que é FDW? (Foreign Data Wrapper)

Foreign Data Wrapper é um recurso (uma extensão) que permite manipular tabelas externas em um banco de dados PostgreSQL, são proxies para alguma outra fonte de dados. Quando você faz uma consulta em uma tabela estrangeira, o FDW consulta a fonte de dados externa e retorna os resultados como se tivessem vindo de uma tabela em seu banco de dados.

O Postgres_FDW funciona como um meio de campo para a conexão entre os bancos de dados, ele faz uma "ponte" para que os dados sejam acessados de outra localidade, através do FDW de cada SGBD. Deste modo, após a configuração do FDW,

é possível integrar um objeto externo (SGBD´s relacionais e não relacionais, assim como arquivos).

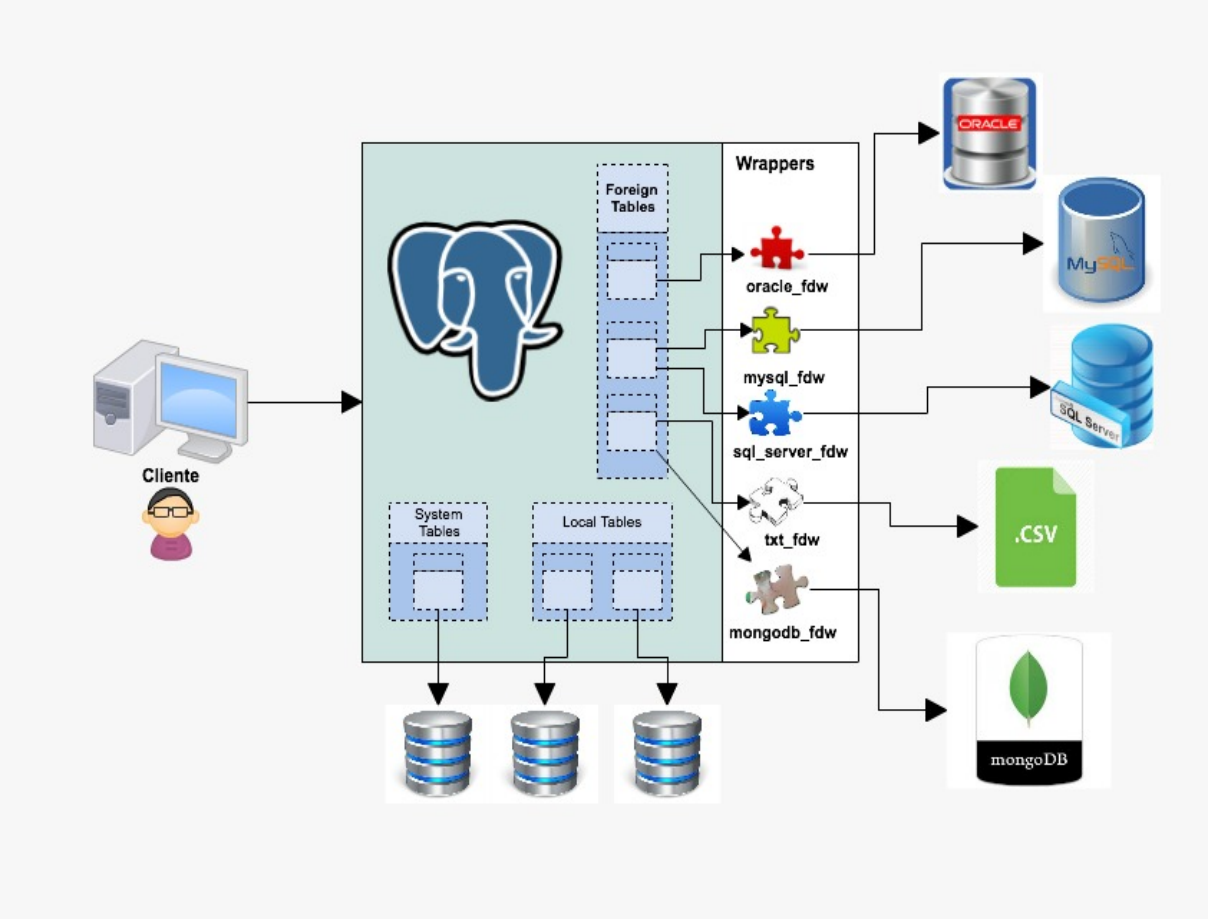

Figura 1. Modelo de funcionamento do FDW.

O FDW não possui apenas a funcionalidade de tornar dados visíveis de várias fontes em um único local, ele também torna capaz a integração do uso de todos esses dados, através de JOINS ou filtros de pesquisa assim como o CRUD de todas essas tabelas externas, possuindo um comportamento de um banco de dados local.

A maioria dos FDWs disponíveis atualmente são Open source ou Software livre, portanto, facilmente é possível encontrar o código de diversas extensões para diferentes aplicações, estando a maioria delas em repositórios no Github, mesmo algumas delas não possuindo licença direta do PostgreSQL. Dependendo da extensão FDW que deseja utilizar, será necessário compilar a extensão antes de utilizá-la.

Além de tudo isso, é importante deixar claro que para utilizar o FDW apenas para leitura de dados, é necessário ter a versão do PostgreSQL 9.1 ou mais. Já em relação a utilização para fazer INSERTS, UPDATES e DELETES, é necessário ter a versão 9.3 ou mais do PostgreSQL.

3. Motivos que fazem o FDW ser uma boa escolha

Atualmente, existem poucas escolhas que possuem características semelhantes às que o FDW pode fornecer aos usuários. A versatilidade que o FDW possui, é uma das principais características que fazem com que a utilização desse recurso seja cada vez mais requisitada. Essa versatilidade vai desde os diferentes tipos que o FDW fornece para seus usuários, como suporte a bancos relacionais, não relacionais(que usam linguagem SQL) e arquivos como JSON,txt, csv e outros tipos.

A única extensão do Postgres que existe nos dias atuais e pode fornecer um serviço parecido ao que o FDW pode oferecer, é o Dblink, porém, ainda assim o FDW apresenta muitas vantagens em relação ao Dblink. Os dois, são muito semelhantes no sentido de serem extensões do Postgre, mesmo assim, o FDW proporciona uma facilidade maior para a manipulação e controle de dados em um banco. Isso acontece, pois o FDW, para o Postgre, é visto como um objeto dentro do banco, ou seja, é como uma tabela dentro do banco de dados, por este motivo, é possível executar chamadas diretas na execução dos comandos.

Em relação a desvantagem que o FDW possui, é possível citar o fator segurança, já que na hora de criar a extensão, é necessário passar a senha de acesso do banco, porém, este problema pode ser resolvido através de métodos de criptografia de dados. O método de criptografia mais utilizado pelas extensões do Postgresql, é o PGcrypto, que geralmente é usado para gerar hash e saltings em bancos de dados que guardam senhas e dando sensíveis.

4. pgAdmin

O pgAdmin é uma ferramenta de gerenciamento para o PostgreSQL, semelhante ao MySQL Workbench, ele pode ser executado como aplicativo web ou desktop. Trata-se de uma interface que facilita o uso do PostgreSQL e suas funcionalidades, assim como suas extensões e ferramentas.

A instalação do pgAdmin pode ser realizada através do site oficial da organização “https://www.pgadmin.org/”, onde é possível encontrar o tutorial adequado para a instalação ocorrer corretamente. Feito isso, podemos analisar a interface do aplicativo e dar os primeiros passos para compreender sua utilização.

Com diversas caixas de diálogo, janelas e botões, se torna intuitivo encontrar as tabelas e bancos de dados criados dentro da aplicação. Para exemplificar, faremos uma apresentação dos passos iniciais dentro da plataforma.

5. Comandos iniciais

Os comandos iniciais do Postgre_SQL são responsáveis por realizar a criação e remoção das tabelas, assim como do próprio banco de dados. Dentre eles estão:

CREATE DATABASE: Cria o banco de dados.

DROP DATABASE: Exclui o banco de dados.

CREATE TABLE: Cria uma tabela.

DROP TABLE: Exclui uma tabela.

IF EXISTS: Comparador de pré-existência de um objeto, normalmente precedido de um DROP TABLE quando se deseja realizar alguns testes locais na criação das tabelas no banco de dados.

INT: Identifica que o atributo será do tipo inteiro.

CHAR: Identifica que o atributo será um caractere.

VARCHAR: Identifica que o atributo será um vetor de caracteres.

NOT NULL: Identifica que o atributo não pode ter valor nulo.

PRIMARY KEY: Identifica que o atributo é uma chave primária, é único na tabela.

Na utilização do pgAdmin, não é possível criar banco de dados diretamente por linhas de comando. Caso esteja sendo utilizado via terminal, após acessar o SGBD, é possível criar um banco com o seguinte comando:

```
CREATE DATABASE nome_do_BD
```

Figura 2. Criando banco de dados via terminal.

Posteriormente à criação do banco de dados, no pgAdmin é necessário abrir um query no banco de dados específico em que será executado os comandos.

Como exemplo de criação de uma tabela,

```
DROP TABLE IF EXISTS funcionarios;

CREATE TABLE funcionarios (
    id_func serial primary key,
    nome varchar(30) NOT NULL,
    sobrenome varchar(30) NOT NULL,
    id_setor int not null
);
```

Figura 3. Exemplo de criação de tabela no pgAdmin.

5.1. Comandos de consultas

Os comandos de consulta no Postgre_SQL são responsáveis por realizar a exibição dos conteúdos presentes nas tabelas, possuem também a funcionalidade de filtrar determinadas informações que se deseja, para organizar o resultado de saída. Alguns exemplos são:

SELECT: Principal comando para a busca de informações nas tabelas.

FROM: Identifica de qual tabela se deseja buscar os dados.

JOIN: Realiza uma junção das informações de duas ou mais tabelas.

WHERE: Realiza uma comparação das informações de duas ou mais tabelas.

Especificamente em relação ao FDW, alguns dos comandos que podem ser uteis, são:

- CREATE EXTENSION
- CREATE, ALTER, DROP SERVER
- CREATE, ALTER, DROP USER MAPPING
- CREATE, ALTER, DROP FOREIGN TABLE

  No último caso, também é muito utilizado o comando IMPORT FOREIGN SCHEMA

6. Configuração do FDW

Para configurar um FDW, são necessários 4 passos básicos,que geralmente são descritos na documentação da extensão FDW que será utilizada (depende de qual SGBD está sendo utilizado) para a importação dos dados. Estes passos são:

1. Instalar/criar a extensão
2. Criar um "Foreign Server"
3. Criar um "User Mapping"(Mapear o usuário)
4. Criar/importar a(s) tabela(s) estrangeiras

Com objetivo de exemplificar os passos descritos acima, estaremos utilizando o postgres_fdw, que é utilizado para o SGBD PostgreSQL. Com esta extensão, é possível criar a extensão utilizando os seguintes comandos.

1º comando: Cria a extensão no banco desejado.

```
create extension postgres_fdw;
```

Figura 4. Exemplo de criação de extensão do FDW.

2º comando: Cria o servidor que o FDW usará. Após o "create server", é escrito o nome do servidor que será criado. Após o "foreign data wrapper", é especificado qual extensão FDW será utilizada, no caso, sempre será a que foi criada na linha de comando anterior. Após "options", é necessário colocar o host, que pode ser especificado como localhost se for um banco de dados que faz parte da mesma rede de dados, ou se for um banco de dados, é possível acessar utilizando o IP da máquina requisitada, dbname seria o nome do banco que está os dados que serão usados, e por fim, a porta de acesso do servidor.

```
create server pg_servidor foreign data wrapper postgres_fdw
 options (host 'localhost', dbname 'nomeDoBD', port '5432');
```

Figura 5. Exemplo de criação do servidor para o FDW.

3º comando: Faz o mapeamento de um usuário para a utilização dos dados. No caso, é necessário colocar o servidor que foi criado no comando acima, e é necessário colocar um usuário e uma senha para o servidor que foi criado. O usuário e a senha geralmente são criados junto com a criação do servidor que fornece os dados ao FDW.

```
create user mapping for postgres server pg_servidor
            options (user 'postgres', password 'senha');
```

Figura 6. Mapeando usuário do FDW no banco relacional.

4º comando:Por último, esse comando faz a importação dos dados através do servidor e torna estes dados públicos para o servidor que está fazendo a requisição através da extensão FDW.

```
import foreign schema public from server
            pg_servidor into public;
```

Figura 7. Importação dos dados do servidor FDW para o banco local.

Note que foram utilizados exatamente os quatro comandos descritos inicialmente, porém, é importante ressaltar que apesar dos diferentes tipos de FDW nos diversos SGBDs que dão suporte a extensão como mysql_fdw e o oracle_fdw, podem haver algumas mudanças, que geralmente não são tão significativas. Na maioria dos casos essas mudanças ocorrem nos options após a criação do servidor, por exemplo, no mysql_fdw, é necessário apenas colocar o host e a porta de acesso, não é necessário colocar o nome do banco que fornecerá os dados.

7. Teste em bancos locais

Após realizar a criação de dois bancos de dados locais, ou seja, ambos foram criados e manipulados através do pgADmin na máquina local (localhost), sendo feito também a criação das tabelas e de seus devidos atributos. Podemos analisar algumas funcionalidades do fdw na prática.

Temos no banco em que foi criada a extensão do postgres_fdw, apenas a tabela “setor”, que possui o id(serial) e nome de cada setor. No outro banco de dados que fazemos a importação dos dados, está a tabela “funcionários”, que possui os campos id_func(id do funcionário), nome, sobrenome, e um id_setor, que indica o setor que esse funcionário trabalha através do id do setor. Foram feitas algumas inserções nestas tabelas, apenas com o objetivo de exemplificar a prática.

	id_setor [PK] integer	nome_setor character varying (30)
1	1	Caixa
2	2	Cozinha
3	3	Açougue

Figura 8. Exibição da tabela setor.

	id_func integer	nome character varying (30)	sobrenome character varying (30)	id_setor integer
1	1	Chris	Greg	1
2	2	Isaac	Newton	2
3	3	André	Schweppes	3
4	4	John	Doe	2
5	5	John	Cena	1

Figura 9. Exibição da tabela funcionários.

Faremos isso com o auxílio do comando SELECT, que será implementado em um banco de dados fazendo uma pesquisa de informações que estão contidas no outro banco. Deste modo é possível analisarmos que o fdw nos trás as informações como se elas fizessem parte do banco local (banco em que a query foi escrita).

Utilizando o seguinte comando:

```
select * from funcionarios natural join setor;
```

Figura 10. Seleção mesclada dos dados das tabelas funcionários e setor.

Obtemos a seguinte resposta do SGBD:

	id_setor integer	id_func integer	nome character varying (30)	sobrenome character varying (30)	nome_setor character varying (30)
1	1	1	Chris	Greg	Caixa
2	2	2	Isaac	Newton	Cozinha
3	3	3	André	Schweppes	Açougue
4	2	4	John	Doe	Cozinha
5	1	5	John	Cena	Caixa

Figura 11. Exibição do resultado da mesclagem.

É possível também, realizarmos as buscas com filtros e restrições, para que sejam apresentados apenas os dados que nos interessam, ou então, que as buscas só retornem algum resultado após encontrar algum conflito de informações.

Utilizando o seguinte comando:

```
Select nome || ' ' || sobrenome As nome From funcionarios
```

Figura 12. Seleção dos atributos nome e sobrenome concatenados da tabela funcionários.

Obtemos a seguinte resposta do SGBD:

	nome text
1	Chris Greg
2	Isaac Newton
3	André Schweppes
4	John Doe
5	John Cena

Figura 13. Exibição do resultado da concatenação.

Não podemos esquecer que a tabela funcionarios está em um banco de dados diferente da tabela em que executamos o comando acima, já que estamos usando estes comando em uma query tool no banco de dados em que se localiza apenas a tabela setor.

O FDW também nos permite ter funcionalidades de CRUD, ou seja, podemos fazer inserção, remoção e alteração de dados. Por exemplo, utilizando o seguinte comando no banco de dados em que apenas temos a tabela setor, podemos excluir um funcionário, que pertence a primeira tabela criada:

```
DELETE FROM funcionarios
WHERE id_func IN (3,5);
```

Figura 14. Exemplo de remoção de tuplas.

Obtemos a seguinte resposta do SGBD:

```
DELETE 2

Query returned successfully in 36 msec.
```

Figura 15. Tempo de resposta do SGBD após remoção realizada com sucesso.

E quando vamos fazer o SELECT para ver como estão os campos, obtemos:

	id_func integer	nome character varying (30)	sobrenome character varying (30)	id_setor integer
1	1	Chris	Greg	1
2	2	Isaac	Newton	2
3	4	John	Doe	2

Figura 16. Exibição da tabela funcionários após a remoção.

Portanto, sim, é possível utilizar o FDW tanto para leitura, quanto para a escrita de dados, seja remotamente ou localmente.

8. Teste em arquivos .CSV

Além da utilização do FDW para bancos relacionais, também pode ser utilizado para alguns tipos de arquivos. Um dos tipos de arquivos que possuem suporte para importação de dados, são os arquivos .CSV, que em geral são utilizados justamente com finalidade de armazenar dados.

A extensão que fornece suporte para arquivos .CSV, é a extensão file_fdw, onde é possível criar a tabela diretamente neste arquivo externo, e posteriormente, manipular os dados que ali foram armazenados. O modo em que este tipo de arquivo pode ser inicializado, é muito parecido com o postgres_fdw, feito no tópico acima, portanto, os passos necessários para começar a importação, é criar a extensão, após isso, criar o servidor, e neste momento vem a diferença, que no caso, é necessário criar tabela e depois definir as propriedades do arquivo, como o formato que os dados irão ser armazenados, nome do arquivo e delimitadores.

Pensando diretamente em um passo a passo, teremos os seguintes comandos:

1- Cria a extensão

```
CREATE EXTENSION file_fdw;
```

Figura 17. Criação da extensão para usar o FDW em arquivos .CSV.

2- Cria um servidor para o FDW

```
CREATE SERVER testfile FOREIGN DATA WRAPPER file_fdw;
```

Figura 18. Criação do servidor.

3- Cria a tabela estrangeira com os campos que estão no arquivo .CSV

```
CREATE FOREIGN TABLE test(
    Nome text,
    Sobrenome text,
    idade int
)
```

Figura 19. Criação da tabela com os campos requeridos.

4- Definir as propriedades do arquivo

```
SERVER testfile(
    FORMAT 'text',
    filename '/test',
    delimiter ':',
    NULL ''
);
```

Figura 20. Exemplo de como definir as propriedades para a extensão file_fdw.

Após estes passos, sua extensão já vai estar rodando, é necessário já ter dados no arquivo .CSV, e é possível fazer o teste se realmente deu certo, apenas iniciando com um comando simples, como um SELECT em um dos campos do arquivo.

9. References

Hanad, Shigeru. (1996).PostgreSQL: Documentação. Disponível em: "https://www.postgresql.org/docs/current/postgres-fdw.html".

Lima, Cleysson. Ribeiro, Alan. (2019) PostgreSQL e FDW: será o fim do dblink? PGConf. Brasil. Disponível em: "https://www.youtube.com/watch?v=ixy03hN2u-U&t=1711s".

Bail, Sam. (2019). PostgresOpen 2019 Intro To Foreign Data Wrappers In Postgres. Disponível em : "https://www.youtube.com/watch?v=Swl0P7cP3-w"

https://www.postgresql.org/

https://4linux.com.br/o-que-e-postgresql/

https://pt.wikipedia.org/wiki/PostgreSQL

https://www.pgadmin.org/