Foreign data what?

Create your own Foreign Data wrapper in PostgreSQL

+

apps associates®

Michael Muehlbeyer

09.10.2025

apps associates®

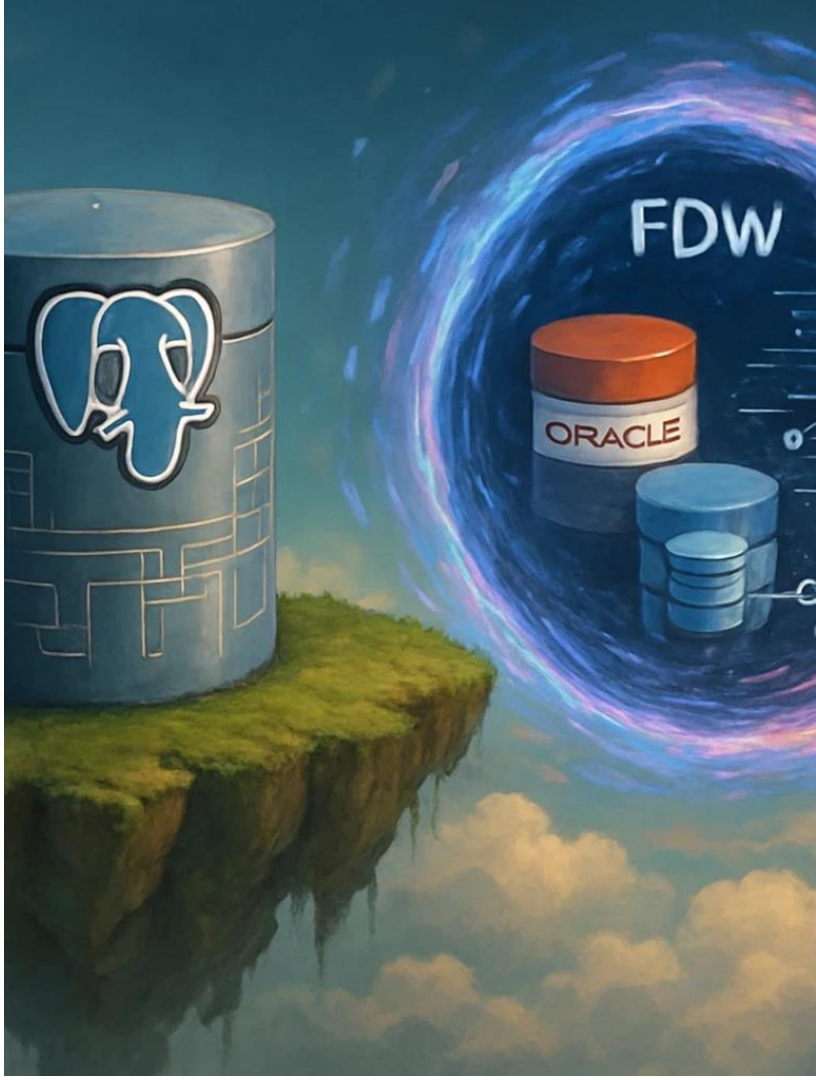
whoami

- Senior Prinicpial Consultant @Apps Associates
- Focssing on data Platforms
  - PostgreSQL, Oracle DB and other RDBMS
  - Kafka & Streaming

- At home in Heilbronn region, BW
- 💻,📚,🐝 keeping, 🎺,🎧

Agenda

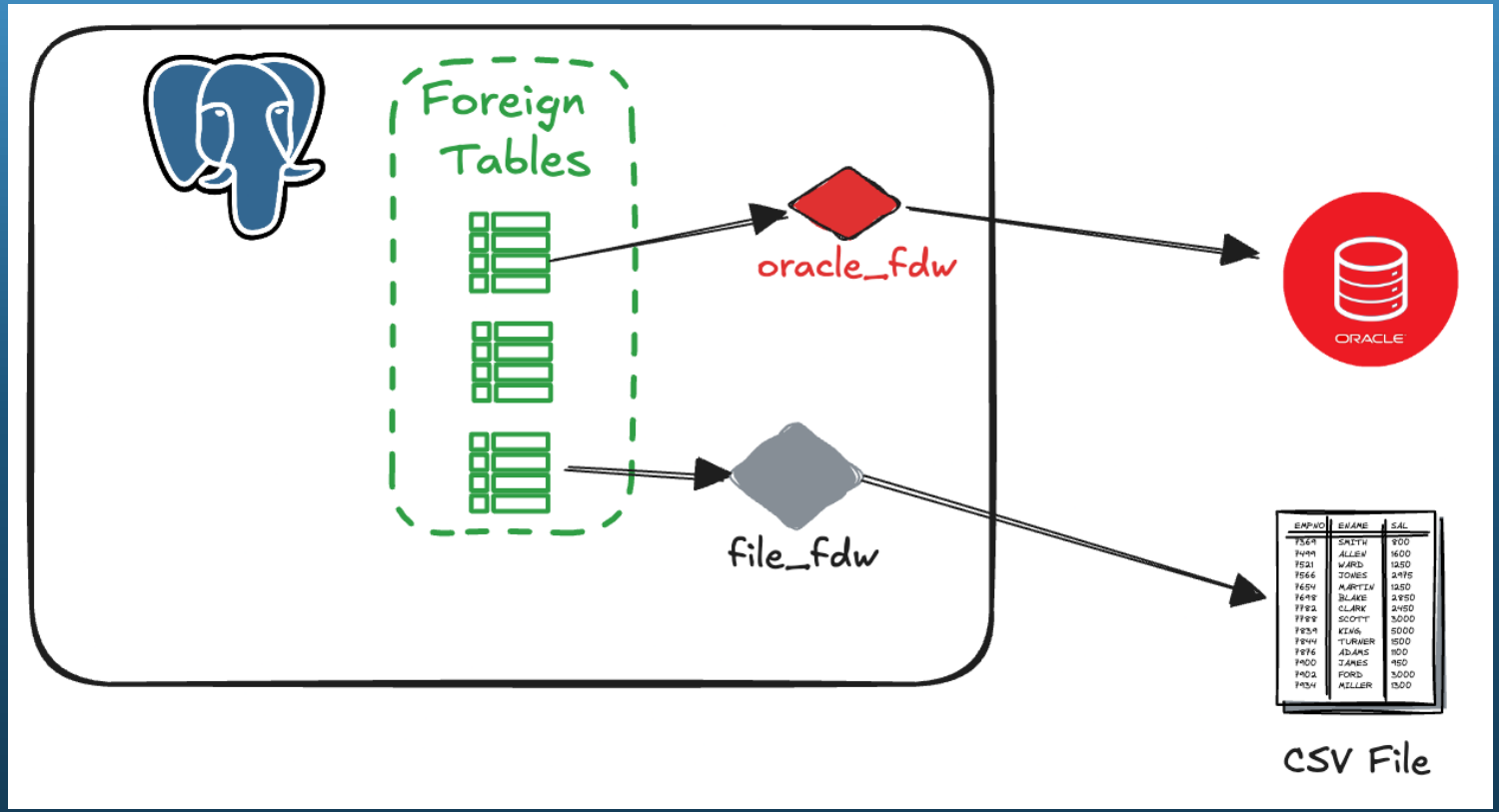
- Overview
- FDW in action
- Sample use cases for FDW
- How to build your own FDW



Overview

- Basically extensions
- Allow you to read/write data from remote/external sources and systems
- Several FDW available to connect/access multiple sources and systems
  - RDBMS
  - Files
  - Streaming
  - ….

Architecture



Overview cont’d

Generic SQL Database Wrappers

Data Source	Type	License	Code	Install	Doc	Notes
ODBC	Native		github			CartoDB took over active development of the ODBC FDW for PG 9.5+
JDBC	Native		github			Not maintained?
JDBC2	Native		github			
JDBC	Native		github		README	More recent than the above, advertises write support.
SQL_Alchemy	Multicorn	PostgreSQL	GitHub	PGXN	documentation	Can be used to access data stored in any database supported by the sqlalchemy python toolkit.
GDAL/OGR	Native	MIT	GitHub	yum.postgresql.org, apt.postgresql.org, and part of PostGIS windows bundle (application stackbuilder)		Can access many kinds of data sources (Relational databases, spreadsheets, CSV files, web feature services, etc). Uses the GDAL library which supports hundreds of formats to access the data. Exposes vector data as PostGIS geometry columns if you have PostGIS installed. Works great with both spatial and non-spatial data.
VirtDB	Native	GPL	GitHub			A generic FDW to access VirtDB data sources (SAP ERP, Oracle RDBMS)
APIs (via Steampipe plugins)	Native	CLI and FDW extension: AGPL 3.0, Plugins: Apache 2.0	CLI on GitHub, FDW extension on GitHub	Steampipe downloads	Steampipe docs, Postgres FDW docs	Steampipe bundles Postgres with an FDW extension that supports a growing ecosystem of plugins. The plugins consume APIs, map them to tables, and enable queries within and across APIs. The plugins are also available as unbundled FDWs for use in any Postgres database.

Specific SQL Database Wrappers

Data Source	Type	License	Code	Install	Doc	Notes
PostgreSQL	Native	PostgreSQL	git.postgresql.org		documentation	
Oracle	Native	PostgreSQL	github	PGXN	website	
MySQL	Native		github	PGXN	example	FDW for MySQL
Informix	Native	PostgreSQL	github			
DB2	Native		github			
Firebird	Native	PostgreSQL	github	PGXN	README	version 1.3.0 released (2022-12)
SQLite	Native	PostgreSQL	github	PGXN	README	An FDW for SQLite3 (write support and several pushdown optimization)
Sybase / MS SQL Server	Native		github	PGXN		An FDW for Sybase and Microsoft SQL server
MonetDB	Native		github			

Overview cont’d

NoSQL Database Wrappers

Data Source	Type	License	Code	Install	Doc	Notes
BigTable or HBase	Native Rust Binding (RPGFFI)	MIT	Github			
Cassandra	Multicorn	MIT	Github	Rankactive		
Cassandra2	Native	MIT	Github			
Cassandra	Multicorn	PostgreSQL	Github			
ClickHouse	Multicorn	BSD	Github		README	
ClickHouse	Native	Apache	Github		README	
ClickHouse	Native		Github		README	
ClickHouse	Wrappers	Apache	GitHub		documentation	A foreign data wrapper for ClickHouse
CouchDB	Native	PostgreSQL	Github	PGXN		Original version
CouchDB	Native	PostgreSQL	Github			golgauth version (9.1 - 9.2+ compatible)
GridDB	Native	PostgreSQL	Github		README	
InfluxDB	Native	PostgreSQL	Github		README	
Kafka	Native	PostgreSQL	GitHub		README	
Kyoto Tycoon	Native	MIT	Github			
MongoDB	Native	GPL3+	Github	PGXN	README	EDB version
MongoDB	Multicorn	MIT	Github			
MongoDB	Multicorn		Github			Yet Another Postgres FDW for MongoDB
Neo4j	Multicorn	GPLv3	Github		README	FWD for Neo4j and also add a Cypher function to Pg
Neo4j	Native	?	Github			
Quasar	Native	Apache	Github			
Redis	Native	PostgreSQL	Github			
Redis	Native	BSD	Github			
RethinkDB	Multicorn	MIT	Github		blog	
Riak	Multicorn	PostgreSQL	Github			
RocksDB	Native	Apache	Github		README	FDW for RocksDB
SPARQL	Multicorn2	PostgreSQL	Codeberg			
WhiteDB	Native	MIT	Github			
RDF Triplestores (SPARQL						A Foreign Data Wrapper to easily access RDF Triplestores via SPARQL, including pushdown of several SQL

Installation

- postgres_fdw
  - Available by default (contrib modules)
  - just create the extension

```
postgres=# SELECT * FROM pg_available_extensions where name='postgres_fdw';
     name     | default_version | installed_version |                       comment
--------------+-----------------+-------------------+-------------------------------------------------------
 postgres_fdw | 1.1             |                   | foreign-data wrapper for remote PostgreSQL servers
```

```
postgres=# create extension postgres_fdw;
CREATE EXTENSION
```

```
postgres=# \dx
                                    List of installed extensions
     Name     | Version | Default version |   Schema   |                   Description
--------------+---------+-----------------+------------+-------------------------------------------------
 plpgsql      | 1.0     | 1.0             | pg_catalog | PL/pgSQL procedural language
 postgres_fdw | 1.1     | 1.1             | public     | foreign-data wrapper for remote PostgreSQL servers
(2 rows)
```

Installation

- oracle_fdw
  - Compile the source

```
[root@postgres02 oracle_fdw]# make install
/usr/bin/mkdir -p '/usr/pgsql-18/lib'
/usr/bin/mkdir -p '/usr/pgsql-18/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-18/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-18/doc/extension'
/usr/bin/install -c -m 755  oracle_fdw.so '/usr/pgsql-18/lib/oracle_fdw.so'
/usr/bin/install -c -m 644 .//oracle_fdw.control '/usr/pgsql-18/share/extension/'
/usr/bin/install -c -m 644 .//oracle_fdw--1.2.sql .//oracle_fdw--1.0--1.1.sql .//oracle_fdw--1.1--1.2.sql  '/usr/pgsql-18/share/extension/'
/usr/bin/install -c -m 644 .//README.oracle_fdw '/usr/pgsql-18/doc/extension/'
```

  - Alternatively, use pgxn
    - **`pgxn install oracle_fdw`**

FDW in action - PostgreSQL/PostgreSQL same or different host

```
chinook=# create extension postgre_fdw;


chinook=# create server remotedemo foreign
data wrapper postgres_fdw options (host
'localhost', dbname 'demo');

chinook=# create user mapping for postgres
server remotedemo options (user 'postgres');

create foreign table airports
( airport_code character(3),
airport_name jsonb,
city jsonb ,
coordinates point
) server remotedemo
OPTIONS (table_name 'airports_data',
schema_name 'bookings');
```

- Load the extension (once per db!)
- Create the foreign server (once per data source)
- Create a user mapping (different users possible
- Create the foreign table (once per table) , import whole schema possible as well

FDW in action - PostgreSQL/Oracle

```
create extension oracle_fdw;


create server ora01 foreign data wrapper oracle_fdw
        options (dbserver '//ora01:1521/freepdb1');

grant usage on foreign server ora01 to postgres;

create user mapping for postgres server ora01 options (user 'CO',
password 'Start123#');



CREATE FOREIGN TABLE customers (
         customer_id        integer  NOT NULL,
         email_address      character varying(255)  NOT NULL,
         full_name          character varying(255)  NOT NULL
      ) SERVER ora01 OPTIONS (schema 'CO', table 'CUSTOMERS');
```

- Load the extension (once per db!)
- Create the foreign server (once per data source)
- Create a user mapping (different users possible
- Create the foreign table (once per table) , import whole schema possible as well

FDW in action – read file

```
create extension file_fdw;



CREATE SERVER file_fdw_test FOREIGN DATA WRAPPER
file_fdw;


CREATE FOREIGN TABLE csv_test (
 id INT,
 message TEXT,
 randnrs BIGINT
)
SERVER file_fdw_test
OPTIONS (
   filename '/tmp/fdw_test.csv',
   format 'csv',
   header 'TRUE'
)
```

- Load the extension
- Create the foreign server
- Create the foreign table

And the optimizer?

- postgres_fdw tries to optimize remote queries to reduce the amount of data transferred
- when a join between foreign tables on the same foreign server happens , the psotgres_fdw sends the entire join to the foreign server
- UPDATE and DELETE queries will be send as a whole to the foreign server

Sample use cases

- Make data accessible from one database to another
- Make data available from external database to PostgreSQL (
  - Oracle
  - MySQL
  - DB2
- Read data from files
  - CSV
  - JSON
  - …

Sample use cases

- Data migrations
  - Legacy systems
  - Version upgrade
  - …
- Access Streaming data
  - Kafka
  - Redis
- Access NoSQL
  - Clickhouse
  - Cassandra

Build your own - FDW

- 2 options around
  - Write your own in C
  - Use Multicorn

- Let‘s check how we can do this for Apache Kafka!

Write your own fdw

- 2 functions
  - Handler
  - Validator (optional)

- The problem
  - I’m not that much in C
  - Complex ☹
  - Not well documented

Multicorn

- PostgreSQL extension to make the FDW development easy
  - “A wrapper of wrappers”
- Allows to use Python
- Last commit to the github repo: 3 years ago ☹ latest release 2016
- Not working right now in several envs

Multicorn2

- Luckily we have vibrant community and chatgpt ☺
  - https://github.com/pgsql-io/multicorn2

```
#download and compile
wget https://github.com/pgsql-io/multicorn2/archive/refs/tags/v3.1.tar.gz
tar -xvf v3.1.tar.gz
cd multicorn2-3.1
make
sudo make install
```

```
Create extension in postgresql

CREATE EXTENSION multicorn;
```

Multicorn2

```
#Create a python module
from multicorn import ForeignDataWrapper
from multicorn.utils import log_to_postgres
from confluent_kafka import Consumer, Producer  # or use another Kafka client library

class KafkaForeignDataWrapper(ForeignDataWrapper):
    def __init__(self, options, columns):
        super(KafkaForeignDataWrapper, self).__init__(options, columns)
        self.columns = columns
        # parse options: bootstrap servers, topic name, group, offset mode, etc.
        self.topic = options.get('topic')
        self.bootstrap_servers = options.get('bootstrap_servers')
        self.group_id = options.get('group_id', None)
        self.starting_offset = options.get('offset', 'beginning')

        # setup Kafka consumer/producer
        self.consumer = Consumer({
            'bootstrap.servers': self.bootstrap_servers,
            'group.id': self.group_id,
            'auto.offset.reset': self.starting_offset
        })
[…]
```

Multicorn2

- Create extension, server and foreign table in PostgreSQL
- Restart you PostgreSQL instance and set $PYTHONPATH before

```
CREATE EXTENSION multicorn;

CREATE SERVER kafka_server FOREIGN DATA WRAPPER multicorn
  OPTIONS (
    wrapper kafka_fdw.ForeignDataWrapper
  );

CREATE FOREIGN TABLE kafka_data (
  key text,
  field1 text,
  field2 integer
) SERVER kafka_server
  OPTIONS (
    topic 'my_topic',
    bootstrap_servers 'kafka:9092',
    group_id 'pgfdw_group',
    offset 'beginning'
  );
```

Multicorn2

- Read data from Kafka

```
SELECT * FROM kafka_data LIMIT 10
```

Further reading

- https://wiki.postgresql.org/wiki/Foreign_data_wrappers
- https://pgpedia.info/f/foreign-data-wrapper-fdw.html
- https://github.com/pgsql-io/multicorn2
- https://github.com/oracle-samples/db-sample-schemas/tree/main/customer_orders

apps associates®