Module X1

Extensions PostgreSQL pour l’utilisateur

24.04

Table des matières

Sur ce document

Formation	Module X1
Titre	Extensions PostgreSQL pour l'utilisateur
Révision	24.04
PDF	https://dali.bo/x1_pdf
EPUB	https://dali.bo/x1_epub
HTML	https://dali.bo/x1_html
Slides	https://dali.bo/x1_slides
TP	https://dali.bo/x1_tp
TP (solutions)	https://dali.bo/x1_solutions

Vous trouverez en ligne les différentes versions complètes de ce document.

Chers lectrices & lecteurs,

Nos formations PostgreSQL sont issues de nombreuses années d'études, d'expérience de terrain et de passion pour les logiciels libres. Pour Dalibo, l'utilisation de PostgreSQL n'est pas une marque d'opportunisme commercial, mais l'expression d'un engagement de longue date. Le choix de l'Open Source est aussi le choix de l'implication dans la communauté du logiciel.

Au-delà du contenu technique en lui-même, notre intention est de transmettre les valeurs qui animent et unissent les développeurs de PostgreSQL depuis toujours : partage, ouverture, transparence, créativité, dynamisme… Le but premier de nos formations est de vous aider à mieux exploiter toute la puissance de PostgreSQL mais nous espérons également qu'elles vous inciteront à devenir un membre actif de la communauté en partageant à votre tour le savoir-faire que vous aurez acquis avec nous.

Nous mettons un point d'honneur à maintenir nos manuels à jour, avec des informations précises et des exemples détaillés. Toutefois malgré nos efforts et nos multiples relectures, il est probable que ce document contienne des oublis, des coquilles, des imprécisions ou des erreurs. Si vous constatez un souci, n'hésitez pas à le signaler via l'adresse formation@dalibo.com[1] !

À propos de DALIBO

DALIBO est le spécialiste français de PostgreSQL. Nous proposons du support, de la formation et du conseil depuis 2005.

Retrouvez toutes nos formations sur https://dalibo.com/formations

[1] mailto:formation@dalibo.com

Remerciements

Ce manuel de formation est une aventure collective qui se transmet au sein de notre société depuis des années. Nous remercions chaleureusement ici toutes les personnes qui ont contribué directement ou indirectement à cet ouvrage, notamment :

Jean-Paul Argudo, Alexandre Anriot, Carole Arnaud, Alexandre Baron, David Bidoc, Sharon Bonan, Franck Boudehen, Arnaud Bruniquel, Pierrick Chovelon, Damien Clochard, Christophe Courtois, Marc Cousin, Gilles Darold, Jehan-Guillaume de Rorthais, Ronan Dunklau, Vik Fearing, Stefan Fercot, Pierre Giraud, Nicolas Gollet, Dimitri Fontaine, Florent Jardin, Virginie Jourdan, Luc Lamarle, Denis Laxalde, Guillaume Lelarge, Alain Lesage, Benoit Lobréau, Jean-Louis Louër, Thibaut Madelaine, Adrien Nayrat, Alexandre Pereira, Flavie Perette, Robin Portigliatti, Thomas Reiss, Maël Rimbault, Julien Rouhaud, Stéphane Schildknecht, Julien Tachoires, Nicolas Thauvin, Be Hai Tran, Christophe Truffier, Cédric Villemain, Thibaud Walkowiak, Frédéric Yhuel.

Forme de ce manuel

Les versions PDF, EPUB ou HTML de ce document sont structurées autour des slides de nos formations. Le texte suivant chaque slide contient le cours et de nombreux détails qui ne peuvent être données à l'oral.

Licence Creative Commons CC-BY-NC-SA

Cela inclut les diapositives, les manuels eux-mêmes et les travaux pratiques. Cette formation peut également contenir quelques images et schémas dont la redistribution est soumise à des licences différentes qui sont alors précisées.

Marques déposées

PostgreSQL® Postgres® et le logo Slonik sont des marques déposées[3] par PostgreSQL Community Association of Canada.

Versions de PostgreSQL couvertes

Ce document ne couvre que les versions supportées de PostgreSQL au moment de sa rédaction, soit les versions 12 à 16.

Sur les versions précédentes susceptibles d'être encore rencontrées en production, seuls quelques points très importants sont évoqués, en plus éventuellement de quelques éléments historiques.

Sauf précision contraire, le système d'exploitation utilisé est Linux.

[3] https://www.postgresql.org/about/policies/trademarks/

1/ Extensions PostgreSQL pour l'utilisateur

1.1 QU'EST-CE QU'UNE EXTENSION ?

- Pour ajouter :
  - types de données
  - méthodes d'indexation
  - fonctions et opérateurs
  - tables, vues…
- Tous sujets, tous publics
- Intégrées (« contribs ») ou projets externes

Les extensions sont un gros point fort de PostgreSQL. Elles permettent de rajouter des fonctionnalités, aussi bien pour les utilisateurs que pour les administrateurs, sur tous les sujets : fonctions utilitaires, types supplémentaires, outils d'administration avancés, voire applications quasi-complètes. Certaines sont intégrées par le projet, mais n'importe qui peut en proposer et en intégrer une.

1.2 ADMINISTRATION DES EXTENSIONS

Techniquement :

- « packages » pour PostgreSQL, en C, SQL, PL/pgSQL…
- Langages : SQL, PL/pgSQL, C (!)…
- Ensemble d'objets livrés ensemble
- contrib <> extension

Une extension est un objet du catalogue, englobant d'autres objets. On peut la comparer à un paquetage Linux.

Une extension peut provenir d'un projet séparé de PostgreSQL (PostGIS, par exemple, ou le *Foreign Data Wrapper* Oracle).

Les extensions les plus simples peuvent se limiter à quelques objets en SQL, certaines sont en PL/pgSQL, beaucoup sont en C. Dans ce dernier cas, il faut être conscient que la stabilité du serveur est encore plus en jeu !

1.2.1 Installation des extensions

- Packagées ou à compiler
- Par base :
  - `CREATE EXTENSION … CASCADE`
  - `ALTER EXTENSION UPDATE`
  - `DROP EXTENSION`
  - `\dx`
- Listées dans `pg_available_extensions`

Au niveau du système d'exploitation, une extension nécessite des objets (binaires, scripts…) dans l'arborescence de PostgreSQL. De nombreuses extensions sont déjà fournies sous forme de paquets dans les distributions courantes ou par le PGDG, ou encore l'outil PGXN. Dans certains cas, il faudra aller sur le site du projet et l'installer soi-même, ce qui peut nécessiter une compilation.

L'extension doit être ensuite déclarée dans chaque base où elle est jugée nécessaire avec `CREATE EXTENSION nom_extension`. Les scripts fournis avec l'extension vont alors créer les objets nécessaires (vues, procédures, tables…). En cas de désinstallation avec `DROP EXTENSION`, ils

seront supprimés. Une extension peut avoir besoin d'autres extensions : l'option `CASCADE` permet de les installer automatiquement.

Le mécanisme couvre aussi la mise à jour des extensions : `ALTER EXTENSION UPDATE` permet de mettre à jour une extension dans PostgreSQL suite à la mise à jour de ses binaires. Cela peut être nécessaire si elle contient des tables à mettre à jour, par exemple. Les versions des extensions disponibles sur le système et celles installées dans la base en cours sont visibles dans la vue `pg_available_extensions`.

Les extensions peuvent être exportées et importées par `pg_dump`/`pg_restore`. Un export par `pg_dump` contient un `CREATE EXTENSION nom_extension`, ce qui permettra de recréer d'éventuelles tables, et le *contenu* de ces tables. Une mise à jour de version majeure, par exemple, permettra donc de migrer les extensions dans leur dernière version installée sur le serveur (changement de prototypes de fonctions, nouvelles vues, etc.).

Sous psql, les extensions présentes dans la base sont visibles avec `\dx` :

```
# \dx
                          Liste des extensions installées
       Nom          | Version |   Schéma   |                    Description
--------------------+---------+------------+-----------------------------------------
 ↪  ----------
 amcheck            | 1.2     | public     | functions for verifying relation integrity
 file_fdw           | 1.0     | public     | foreign-data wrapper for flat file access
 hstore             | 1.6     | public     | data type for storing sets of (key,
  ↪  value) pairs
 pageinspect        | 1.9     | public     | inspect the contents of database pages
  ↪  at...
 pg_buffercache     | 1.3     | public     | examine the shared buffer cache
 pg_prewarm         | 1.2     | public     | prewarm relation data
 pg_rational        | 0.0.1   | public     | bigint fractions
 pg_stat_statements | 1.10    | public     | track execution statistics of all SQL
  ↪  statements...
 plpgsql            | 1.0     | pg_catalog | PL/pgSQL procedural language
 plpython3u         | 1.0     | pg_catalog | PL/Python3U untrusted procedural language
 postgres_fdw       | 1.0     | public     | foreign-data wrapper for remote
  ↪  PostgreSQL servers
 unaccent           | 1.1     | public     | text search dictionary that removes
  ↪  accents
```

1.3 CONTRIBS - FONCTIONNALITÉS

- Livrées avec le code source de PostgreSQL
- Habituellement packagées (`postgresql-*-contrib`)
- De qualité garantie car maintenues par le projet
- Optionnelles, désactivées par défaut
- Ou en cours de stabilisation
- Documentées : Chapitre F : « Modules supplémentaires fournis »[1]

Une « contrib » est habituellement une extension, sauf quelques exceptions qui ne créent pas d'objets de catalogue (`auto_explain` par exemple). Elles sont fournies directement dans l'arborescence de PostgreSQL, et suivent donc strictement son rythme de révision. Leur compatibilité est ainsi garantie. Les distributions les proposent parfois dans des paquets séparés (`postgresql-contrib-9.6` , `postgresql14-contrib` …), dont l'installation est fortement conseillée.

Il s'agit soit de fonctionnalités qui n'intéressent pas tout le monde (`hstore` , `uuid` , `pg_trgm` , `pgstattuple` …), ou en cours de stabilisation (comme l'autovacuum avant PostgreSQL 8.1), ou à l'inverse de dépréciation (`xml2`).

La documentation des contribs est dans le chapitre F des annexes[2], et est donc fréquemment oubliée par les nouveaux utilisateurs.

[2] https://docs.postgresql.fr/current/contrib.html

1.4 QUELQUES EXTENSIONS

…plus ou moins connues

1.4.1 pgcrypto

Module contrib de chiffrement :

- Nombreuses fonctions pour chiffrer et déchiffrer des données
- Gros inconvénient : oubliez les index sur les données chiffrées !
- N'oubliez pas de chiffrer la connexion (SSL)
- Permet d'avoir une seule méthode de chiffrement pour tout ce qui accède à la base

Fourni avec PostgreSQL, vous permet de chiffrer vos données[3] :

- directement ;
- avec une clé PGP (gérée par exemple avec GnuPG), ce qui est préférable ;
- selon divers algorithmes courants ;
- différemment selon chaque ligne/champ.

Voici un exemple de code:

```
CREATE EXTENSION pgcrypto;
UPDATE utilisateurs SET mdp = crypt('mon nouveau mot de passe',gen_salt('md5'));
INSERT INTO table_secrete (encrypted)
VALUES (pgp_sym_encrypt('mon secret','motdepasse'));
```

L'appel à `gen_salt` permet de rajouter une partie aléatoire à la chaîne à chiffrer, ce qui évite que la même chaîne chiffrée deux fois retourne le même résultat. Cela limite donc les attaques par dictionnaire.

La base effectuant le (dé)chiffrement, cela évite certains allers-retours. Il est préférable que la clé de déchiffrement ne soit pas *dans* l'instance, et soit connue et fournie par l'applicatif. La communication avec cet applicatif doit être sécurisée par SSL pour que les clés et données ne transitent pas en clair.

Un gros inconvénient des données chiffrées dans la table est l'impossibilité complète de les indexer, même avec un index fonctionnel : les données déchiffrées seraient en clair dans cet index ! Une recherche implique donc de parcourir et déchiffrer chaque ligne…

[3] https://docs.postgresql.fr/current/pgcrypto.html

1.4.2 hstore : stockage clé/valeur

- Contrib
- Type `hstore`
- Stockage clé-valeur
- Plus simple que JSON

```
INSERT INTO demo_hstore (meta) VALUES ('river=>t');
SELECT * FROM demo_hstore WHERE meta@>'river=>t';
```

`hstore` fournit un type très simple pour stocker des clés/valeur :

```
CREATE EXTENSION hstore ;

CREATE TABLE demo_hstore(id serial, meta hstore);
INSERT INTO demo_hstore (meta) VALUES ('river=>t');
INSERT INTO demo_hstore (meta) VALUES ('road=>t,secondary=>t');
INSERT INTO demo_hstore (meta) VALUES ('road=>t,primary=>t');
CREATE INDEX idxhstore ON demo_hstore USING gist (meta);

SELECT * FROM demo_hstore WHERE meta@>'river=>t';

 id |     meta
----+--------------
 15 | "river"=>"t"
```

Cette extension a rendu, et rend encore, bien des services. Cependant le type JSON (avec le type binaire `jsonb`) est généralement préféré.

1.4.3 PostgreSQL Anonymizer

- Extension externe (Dalibo)
- Masquage statique et dynamique
- Export anonyme (`pg_dump_anon`)
- Les règles de masquage sont écrites en SQL
- Autodétection de colonnes identifiantes
- Plus simple et plus sûr qu'un ETL

Postgresql Anonymizer[4] est une extension pour masquer ou remplacer les données personnelles[5] dans une base PostgreSQL. Elle est développée par Damien Clochard de Dalibo.

Le projet fonctionne selon une **approche déclarative**, c'est à dire que les règles de masquage[6] sont déclarées directement dans le modèle de données avec des ordres DDL.

Une fois que les règles de masquage sont définies, on peut accéder aux données masquées de 3 façons différentes :

- export anonyme[7] : extraire les données masquées dans un fichier SQL ;
- masquage statique[8] : supprimer une fois pour toutes les données personnelles ;
- masquage dynamique[9] : cacher les données personnelles seulement pour les utilisateurs masqués.

Par ailleurs, l'extension fournit toute une gamme de fonctions de masquage[10] : randomisation, génération de données factices, destruction partielle, brassage, ajout de bruit, etc. On peut également écrire ses propres fonctions de masquage !

Au-delà du masquage, il est également possible d'utiliser une autre approche appelée généralisation[11] qui est bien adaptée pour les statistiques et l'analyse de données.

Enfin, l'extension offre un panel de fonctions de détection[12] qui tentent de deviner quelles colonnes doivent être anonymisées.

Un module de formation lui est consacré[13].

Exemple :

```
=# SELECT * FROM people;

 id | firstname | lastname |   phone
----+-----------+----------+------------
 T1 | Sarah     | Conor    | 0609110911
```

Étape 1 : activer le masquage dynamique

```
=# CREATE EXTENSION IF NOT EXISTS anon CASCADE;
=# SELECT anon.start_dynamic_masking();
```

Étape 2 : déclarer un utilisateur masqué

```
=# CREATE ROLE skynet LOGIN;
=# SECURITY LABEL FOR anon ON ROLE skynet IS 'MASKED';
```

Étape 3 : déclarer les règles de masquage

[4] https://postgresql-anonymizer.readthedocs.io/en/stable/
[5] https://en.wikipedia.org/wiki/Personally_identifiable_information
[6] https://postgresql-anonymizer.readthedocs.io/en/stable/declare_masking_rules/
[7] https://postgresql-anonymizer.readthedocs.io/en/stable/anonymous_dumps/
[8] https://postgresql-anonymizer.readthedocs.io/en/latest/static_masking/
[9] https://postgresql-anonymizer.readthedocs.io/en/stable/dynamic_masking/
[10] https://postgresql-anonymizer.readthedocs.io/en/latest/masking_functions/
[11] https://postgresql-anonymizer.readthedocs.io/en/stable/generalization/
[12] https://postgresql-anonymizer.readthedocs.io/en/stable/detection/
[13] https://dali.bo/y5_html

```
=# SECURITY LABEL FOR anon ON COLUMN people.lastname
-# IS 'MASKED WITH FUNCTION anon.fake_last_name()';

=# SECURITY LABEL FOR anon ON COLUMN people.phone
-# IS 'MASKED WITH FUNCTION anon.partial(phone,2,$$******$$,2)';
```

Étape 4 : se connecter avec l'utilisateur masqué

```
=# \c - skynet
=# SELECT * FROM people;

 id | firstname | lastname  |   phone
----+-----------+-----------+------------
 T1 | Sarah     | Stranahan | 06******11
```

1.4.4 PostGIS

- Projet indépendant, GPL, https://postgis.net/
- Module spatial pour PostgreSQL
  - Extension pour types géométriques/géographiques & outils
  - La référence des bases de données spatiales
  - « quelles sont les routes qui coupent le Rhône ? »
  - « quelles sont les villes adjacentes à Toulouse ? »
  - « quels sont les restaurants situés à moins de 3 km de la Nationale 12 ? »

PostGIS ajoute le support d'objets géographiques à PostgreSQL. C'est un projet totalement indépendant développé par la société Refractions Research sous licence GPL, soutenu par une communauté

active, utilisée par des spécialistes du domaine géospatial (IGN, BRGM, AirBNB, Mappy, Openstreetmap, Agence de l'eau…), mais qui peut convenir pour des projets plus modestes.

Techniquement, c'est une extension transformant PostgreSQL en serveur de données spatiales, qui sera utilisé par un Système d'Information Géographique (SIG), tout comme le SDE de la société ESRI ou bien l'extension Oracle Spatial. PostGIS se conforme aux directives du consortium OpenGIS et a été certifié par cet organisme comme tel, ce qui est la garantie du respect des standards par PostGIS.

PostGIS permet d'écrire des requêtes de ce type :

```
SELECT restaurants.geom, restaurants.name FROM restaurants
  WHERE EXISTS (SELECT 1 FROM routes
                  WHERE ST_DWithin(restaurants.geom, routes.geom, 3000)
                AND route.name = 'Nationale 12')
```

PostGIS fournit les fonctions d'indexation qui permettent d'accéder rapidement aux objets géométriques, au moyen d'index GiST. La requête ci-dessus n'a évidemment pas besoin de parcourir tous les restaurants à la recherche de ceux correspondant aux critères de recherche.

La liste des fonctionnalités comprend le support des coordonnées géodésiques ; des projections et reprojections dans divers systèmes de coordonnées locaux (Lambert93 en France par exemple) ; des opérateurs d'analyse géométrique (enveloppe convexe, simplification…)

PostGIS est intégré aux principaux serveurs de carte, ETL, et outils de manipulation.

La version 3.0 apporte la gestion du parallélisme, un meilleur support de l'indexation SP-GiST et GiST, ainsi qu'un meilleur support du type GeoJSON.

1.4.5 Mais encore…

- **uuid-ossp** : gérer des UUID
- **unaccent** : supprime des accents
- **citex** : recherche insensible à la casse

1.4.6 Autres extensions connues

- Compatibilité :
  - orafce
- Extensions propriétaires évitant un *fork* :
  - Citus (*sharding*)
  - TimescaleDB (*time series*)
  - être sûr que PostgreSQL a atteint ses limites !

Les extensions permettent de diffuser des bibliothèques de fonction pour la compatibilité avec du code d'autres produits : orafce est un exemple bien connu.

Pour éviter de maintenir un *fork* complet de PostgreSQL, certains éditeurs offrent leur produit sous forme d'extension, souvent avec une version communautaire intégrant les principales fonctionnalités. Par exemple :

- Citus permet du *sharding* ;
- TimescaleDB gère les séries temporelles.

Face à des extensions extérieures, on gardera à l'esprit qu'il s'agit d'un produit supplémentaire à maîtriser et administrer, et l'on cherchera d'abord à tirer le maximum du PostgreSQL communautaire.

1.5 EXTENSIONS POUR DE NOUVEAUX LANGAGES

- PL/pgSQL par défaut
- Ajouter des langages :
  - PL/python
  - PL/perl
  - PL/lua
  - PL/sh
  - PL/R
  - PL/Java
  - etc.

SQL et PL/pgSQL ne sont pas les seuls langages utilisables au niveau d'un serveur PostgreSQL. PL/pgSQL est installé par défaut en tant qu'extension. Il est possible de rajouter les langages python, perl, R, etc. et de coder des fonctions dans ces langages. Ces langages ne sont pas fournis par l'installation standard de PostgreSQL. Une installation via les paquets du système d'exploitation est sans doute le plus simple.

1.6 ACCÈS DISTANTS

Accès à des bases distantes

- Contribs :
  - dblink (ancien)
  - les *foreign data wrappers* : postgresql_fdw, mysql_fdw…
- Sharding :
  - PL/Proxy
  - Citus

Les accès distants à d'autres bases de données sont généralement disponibles par des extensions. L'extension dblink permet d'accéder à une autre instance PostgreSQL mais elle est ancienne, et l'on préférera le *foreign data wrapper* postgresql_fdw, disponible dans les contribs. D'autres FDW sont des projets extérieurs : ora_fdw, mysql_fdw, etc.

Une solution de *sharding* n'est pas encore intégrée à PostgreSQL mais des outils existent : PL/Proxy fournit des fonctions pour répartir des accès mais implique de refondre le code. Citus est une extension plus récente et plus transparente.

1.7 CONTRIBS ORIENTÉS DBA

Accès à des informations ou des fonctions de bas niveau :

- **pg_prewarm** : sauvegarde & restauration de l'état du cache de la base
- **pg_buffercache** : état du cache
- **pgstattuple** (fragmentation des tables et index), **pg_freespacemap** (blocs libres), **pg_visibility** (*visibility map*)
- **pageinspect** : inspection du contenu d'une page
- **pgrowlocks** : informations détaillées sur les enregistrements verrouillés
- **pg_stat_statement** (requêtes normalisées), **auto_explain** (plans)
- **amcheck** : validation des index
- … et de nombreux projets externes

Tous ces modules permettent de manipuler une facette de PostgreSQL à laquelle on n'a normalement pas accès. Leur utilisation est parfois très spécialisée et pointue.

En plus des contribs listés ci-dessus, de nombreux projets externes existent : toastinfo, pg_stat_kcache, pg_qualstats, PoWa, pg_wait_sampling, hypopg…

Pour plus de détails, consulter les modules X2[14] et X3[15].

[14] https://dali.bo/x2_html
[15] https://dali.bo/x3_html

1.8 PGXN

PostgreSQL eXtension Network :

- Site web : pgxn.org[16]
  - nombreuses extensions
  - volontariat
  - aucune garantie de qualité
  - tests soigneux requis
- Et optionnellement client en python pour automatisation de déploiement
- Ancêtre : pgFoundry
- Beaucoup de projets sont aussi sur github

Le site PGXN fournit une vitrine à de nombreux projets gravitant autour de PostgreSQL.

PGXN a de nombreux avantages, dont celui de demander aux projets participants de respecter un certain cahier des charges permettant l'installation automatisée des modules hébergés. Ceci peut par exemple être réalisé avec le client `pgxn` fourni :

```
> pgxn search --dist fdw
multicdr_fdw 1.2.2
    MultiCDR *FDW* ================== Foreign Data Wrapper for representing
    CDR files stream as an external SQL table. CDR files from a directory
    can be read into a table with a specified field-to-column...

redis_fdw 1.0.0
    Redis *FDW* for PostgreSQL 9.1+ ============================== This
    PostgreSQL extension implements a Foreign Data Wrapper (*FDW*) for the
    Redis key/value database: http://redis.io/ This code is...

jdbc_fdw 1.0.0
    Also,since the JVM being used in jdbc *fdw* is created only once for the
    entire psql session,therefore,the first query issued that uses jdbc
    +fdw* shall set the value of maximum heap size of the JVM(if...

mysql_fdw 2.1.2
    ... This PostgreSQL extension implements a Foreign Data Wrapper (*FDW*)
    for [MySQL][1]. Please note that this version of mysql_fdw only works
    with PostgreSQL Version 9.3 and greater, for previous version...

www_fdw 0.1.8
    ... library contains a PostgreSQL extension, a Foreign Data Wrapper
    (*FDW*) handler of PostgreSQL which provides easy way for interacting
    with different web-services.

mongo_fdw 2.0.0
    MongoDB *FDW* for PostgreSQL 9.2 ============================== This
    PostgreSQL extension implements a Foreign Data Wrapper (*FDW*) for
```

```
    MongoDB.

firebird_fdw 0.1.0
    ...  -
    http://www.postgresql.org/docs/current/interactive/postgres-*fdw*.html *
    Other FDWs  - https://wiki.postgresql.org/wiki/*Fdw*  -
    http://pgxn.org/tag/*fdw*/

json_fdw 1.0.0
    ... This PostgreSQL extension implements a Foreign Data Wrapper (*FDW*)
    for JSON files. The extension doesn't require any data to be loaded into
    the database, and supports analytic queries against array...

postgres_fdw 1.0.0
    This port provides a read-only Postgres *FDW* to PostgreSQL servers in
    the 9.2 series. It is a port of the official postgres_fdw contrib module
    available in PostgreSQL version 9.3 and later.

osm_fdw 3.0.0
    ... "Openstreetmap pbf foreign data wrapper") (*FDW*) for reading
    [Openstreetmap PBF](http://wiki.openstreetmap.org/wiki/PBF_Format
    "Openstreetmap PBF") file format (*.osm.pbf) ## Requirements  *...

odbc_fdw 0.1.0
    ODBC *FDW* (beta) for PostgreSQL 9.1+
    ================================== This PostgreSQL extension implements
    a Foreign Data Wrapper (*FDW*) for remote databases using Open Database
    Connectivity(ODBC)...

couchdb_fdw 0.1.0
    CouchDB *FDW* (beta) for PostgreSQL 9.1+
    ===================================== This PostgreSQL extension
    implements a Foreign Data Wrapper (*FDW*) for the CouchDB document-
    oriented database...

treasuredata_fdw 1.2.14
    ## INSERT INTO statement This *FDW* supports `INSERT INTO` statement.
    With `atomic_import` is `false`, the *FDW* imports INSERTed rows as
    follows.

twitter_fdw 1.1.1
    Installation ------------  $ make && make install  $ psql -c "CREATE
    EXTENSION twitter_fdw" db The CREATE EXTENSION statement creates not
    only *FDW* handlers but also Data Wrapper, Foreign Server, User...

ldap_fdw 0.1.1
    ... is an initial working on a PostgreSQL's Foreign Data Wrapper (*FDW*)
    to query LDAP servers. By all means use it, but do so entirely at your
    own risk! You have been warned! Do you like to use it in...

git_fdw 1.0.2
    # PostgreSQL Git Foreign Data Wrapper [![Build Status](https://travis-
    ci.org/franckverrot/git_fdw.svg?branch=master)](https://travis-
    ci.org/franckverrot/git_fdw) git\_fdw is a Git Foreign Data...

oracle_fdw 2.0.0
```

```
    Foreign Data Wrapper for Oracle ==============================
    oracle_fdw is a PostgreSQL extension that provides a Foreign Data
    Wrapper for easy and efficient access to Oracle databases, including...

foreign_table_exposer 1.0.0
    # foreign_table_exposer This PostgreSQL extension exposes foreign tables
    like a normal table with rewriting Query tree. Some BI tools can't
    detect foreign tables since they don't consider them when...

cstore_fdw 1.6.0
    cstore_fdw ========== [![Build Status](https://travis-
    ci.org/citusdata/cstore_fdw.svg?branch=master)][status] [![Coverage](htt
    p://img.shields.io/coveralls/citusdata/cstore_fdw/master.svg)][coverage]
    ...

multicorn 1.3.5
    [![PGXN version](https://badge.fury.io/pg/multicorn.svg)](https://badge.
    fury.io/pg/multicorn) [![Build
    Status](https://jenkins.dalibo.info/buildStatus/public/Multicorn)]()
    Multicorn =========...

tds_fdw 1.0.7
    # TDS Foreign data wrapper * **Author:** Geoff Montee * **Name:**
    tds_fdw * **File:** tds_fdw/README.md ## About This is a [PostgreSQL
    foreign data...

pmpp 1.2.3
    ... Having foreign server definitions and user mappings makes for
    cleaner function invocations.

file_textarray_fdw 1.0.1
    ### File Text Array Foreign Data Wrapper for PostgreSQL This *FDW* is
    similar to the provided file_fdw, except that instead of the foreign
    table having named fields to match the fields in the data...

floatfile 1.3.0
    Also I'd need to compare the performance of this vs an *FDW*. If I do
    switch to an *FDW*, I'll probably use [Andrew Dunstan's
    `file_text_array_fdw`](https://github.com/adunstan/file_text_array_fdw)
    as a...

pg_pathman 1.4.13
    ... event handling;  * Non-blocking concurrent table partitioning;  *
    +FDW* support (foreign partitions);  * Various GUC toggles and
    configurable settings.
```

Pour peu que le Instant Client d'Oracle soit installé, on peut par exemple lancer :

```
> pgxn install oracle_fdw
INFO: best version: oracle_fdw 1.1.0
INFO: saving /tmp/tmpihaor2is/oracle_fdw-1.1.0.zip
INFO: unpacking: /tmp/tmpihaor2is/oracle_fdw-1.1.0.zip
INFO: building extension
gcc -O3 -O0 -Wall -Wmissing-prototypes -Wpointer-arith [...]
[...]
INFO: installing extension
```

```
/usr/bin/mkdir -p '/opt/postgres/lib'
/usr/bin/mkdir -p '/opt/postgres/share/extension'
/usr/bin/mkdir -p '/opt/postgres/share/extension'
/usr/bin/mkdir -p '/opt/postgres/share/doc/extension'
/usr/bin/install -c -m 755  oracle_fdw.so '/opt/postgres/lib/oracle_fdw.so'
/usr/bin/install -c -m 644 oracle_fdw.control '/opt/postgres/share/extension/'
/usr/bin/install -c -m 644 oracle_fdw--1.1.sql\oracle_fdw--1.0--1.1.sql
                           '/opt/postgres/share/extension/'
/usr/bin/install -c -m 644 README.oracle_fdw \
                           '/opt/postgres/share/doc/extension/'
```

Attention : le fait qu'un projet soit hébergé sur PGXN n'est absolument pas une validation de la part du projet PostgreSQL. De nombreux projets hébergés sur PGXN sont encore en phase de développement, voire abandonnés. Il faut avoir le même recul que pour n'importe quel autre brique libre.

1.9 CRÉER SON EXTENSION

- Pas si compliqué
- Peut-être juste quelques fonctions SQL
- Référence : documentation, *Empaqueter des objets dans une extension*[17]
- Exemples SQL et C : blog Dalibo[18]

Il n'est pas très compliqué de créer sa propre extension pour diffuser aisément des outils. Elle peut se limiter à des fonctions en SQL ou PL/pgSQL. Le versionnement des extensions et la facilité de mise à jour peuvent être extrêmement utiles.

Deux exemples de création de fonctions en SQL ou C sont disponibles sur le blog Dalibo[19]. Un autre billet de blog présente une extension utilisable pour l'archivage[20].

La référence reste évidemment la documentation de PostgreSQL, chapitre *Empaqueter des objets dans une extension*[21].

[19] https://blog.dalibo.com/2023/06/08/hackingpg1.html
[20] https://blog.dalibo.com/2023/07/28/hackingpg2.html
[21] https://docs.postgresql.fr/current/extend-extensions.html

1.10 CONCLUSION

- Un nombre toujours plus important d'extension pour étendre les possibilités de PostgreSQL
- Un site central pour les extensions : PGXN.org
- Rajoutez les vôtres !

Cette possibilité d'étendre les fonctionnalités de PostgreSQL est vraiment un atout majeur du projet PostgreSQL. Cela permet de tester des fonctionnalités sans avoir à toucher au moteur de PostgreSQL et risquer des états instables.

Une fois l'extension mature, elle peut être intégrée directement dans le code de PostgreSQL si elle est considérée utile au moteur.

N'hésitez pas à créer vos propres extensions et à les diffuser !

1.10.1 Questions

N'hésitez pas, c'est le moment !

1.11 TRAVAUX PRATIQUES

1.11.1 Masquage statique de données avec PostgreSQL Anonymizer

But : Découverte de l'extension PostgreSQL Anonymizer et du masquage statique

Installer l'extension PostgreSQL Anonymizer en suivant la procédure décrite sur la page Installation[a] de la documentation.

[a] https://postgresql-anonymizer.readthedocs.io/en/stable/INSTALL/

Créer une table `customer` :

```
CREATE TABLE customer (
    id SERIAL PRIMARY KEY,
    firstname TEXT,
    lastname TEXT,
    phone TEXT,
    birth DATE,
    postcode TEXT
);
```

Ajouter des individus dans la table :

```
INSERT INTO customer
VALUES
(107,'Sarah','Conor','060-911-0911', '1965-10-10', '90016'),
(258,'Luke', 'Skywalker', NULL, '1951-09-25', '90120'),
(341,'Don', 'Draper','347-515-3423', '1926-06-01', '04520')
;
```

Lire la documentation sur comment déclarer une règle de masquage[a] et placer une règle pour générer un faux nom de famille sur la colonne `lastname`. Déclarer une règle de masquage statique sur la colonne `lastname` et l'appliquer. Vérifier le contenu de la table.

[a] https://postgresql-anonymizer.readthedocs.io/en/latest/declare_masking_rules/

Réappliquer le masquage statique[a]. Qu'observez-vous ?

[a] https://postgresql-anonymizer.readthedocs.io/en/latest/static_masking/

1.11.2 Masquage dynamique de données avec PostgreSQL Anonymizer

But : Mettre en place un masquage dynamique avec PostgreSQL Anonymizer

Parcourir la liste des fonctions de masquage[a] et écrire une règle pour cacher partiellement le numéro de téléphone. Activer le masquage dynamique. Appliquer le masquage dynamique uniquement sur la colonne `phone` pour un nouvel utilisateur nommé **soustraitant**.

[a] https://postgresql-anonymizer.readthedocs.io/en/latest/masking_functions/

1.11.3 Masquage statique de données avec PostgreSQL Anonymizer

Installer l'extension PostgreSQL Anonymizer en suivant la procédure décrite sur la page Installation[a] de la documentation.

[a] https://postgresql-anonymizer.readthedocs.io/en/stable/INSTALL/

Sur Rocky Linux ou autre dérivé Red Hat, depuis les dépôts du PGDG :

```
sudo dnf install postgresql_anonymizer_14
```

Au besoin, remplacer 14 par la version de l'instance PostgreSQL.

La base de travail ici se nomme **sensible**. Se connecter à l'instance pour initialiser l'extension :

```
ALTER DATABASE sensible SET session_preload_libraries = 'anon' ;
```

Après reconnexion à la base **sensible** :

```
CREATE EXTENSION anon CASCADE;

SELECT anon.init();  -- ne pas oublier !
```

Créer une table `customer` :

```
CREATE TABLE customer (
    id SERIAL PRIMARY KEY,
    firstname TEXT,
    lastname TEXT,
    phone TEXT,
    birth DATE,
    postcode TEXT
);
```

Ajouter des individus dans la table :

```
INSERT INTO customer
VALUES
(107,'Sarah','Conor','060-911-0911', '1965-10-10', '90016'),
(258,'Luke', 'Skywalker', NULL, '1951-09-25', '90120'),
(341,'Don', 'Draper','347-515-3423', '1926-06-01', '04520')
;

SELECT * FROM customer ;
```

```
 id  | firstname | lastname  |    phone     |   birth    | postcode
-----+-----------+-----------+--------------+------------+----------
 107 | Sarah     | Conor     | 060-911-0911 | 1965-10-10 | 90016
 258 | Luke      | Skywalker |              | 1951-09-25 | 90120
 341 | Don       | Draper    | 347-515-3423 | 1926-06-01 | 04520
```

> Lire la documentation sur comment déclarer une règle de masquage[a] et placer une règle pour générer un faux nom de famille sur la colonne `lastname`. Déclarer une règle de masquage statique sur la colonne `lastname` et l'appliquer. Vérifier le contenu de la table.
>
> [a] https://postgresql-anonymizer.readthedocs.io/en/latest/declare_masking_rules/

```
SECURITY LABEL FOR anon ON COLUMN customer.lastname
IS 'MASKED WITH FUNCTION anon.fake_last_name()' ;
```

Si on consulte la table avec :

```
SELECT * FROM customer ;
```

les données ne sont pas encore masquées car la règle n'est pas appliquée. L'application se fait avec :

```
SELECT anon.anonymize_table('customer') ;

SELECT * FROM customer;
```

```
 id  | firstname | lastname  |    phone     |   birth    | postcode
-----+-----------+-----------+--------------+------------+----------
 107 | Sarah     | Waelchi   | 060-911-0911 | 1965-10-10 | 90016
 258 | Luke      | Lemke     |              | 1951-09-25 | 90120
 341 | Don       | Shanahan  | 347-515-3423 | 1926-06-01 | 04520
```

NB : les données de la table ont ici bien été modifiées sur le disque.

> Réappliquer le masquage statique[a]. Qu'observez-vous ?
>
> [a] https://postgresql-anonymizer.readthedocs.io/en/latest/static_masking/

Si l'on relance l'anonymisation plusieurs fois, les données factices vont changer car la fonction `fake_last_name()` renvoie des valeurs différentes à chaque appel.

```
SELECT anon.anonymize_table('customer');

SELECT * FROM customer;
```

```
 id  | firstname | lastname  |    phone     |   birth    | postcode
-----+-----------+-----------+--------------+------------+----------
 107 | Sarah     | Smith     | 060-911-0911 | 1965-10-10 | 90016
 258 | Luke      | Sanford   |              | 1951-09-25 | 90120
 341 | Don       | Goldner   | 347-515-3423 | 1926-06-01 | 04520
```

1.11.4 Masquage dynamique de données avec PostgreSQL Anonymizer

> Parcourir la liste des fonctions de masquage[a] et écrire une règle pour cacher partiellement le numéro de téléphone. Activer le masquage dynamique. Appliquer le masquage dynamique uniquement sur la colonne `phone` pour un nouvel utilisateur nommé **soustraitant**.
>
> [a] https://postgresql-anonymizer.readthedocs.io/en/latest/masking_functions/

```
SELECT anon.start_dynamic_masking();

SECURITY LABEL FOR anon ON COLUMN customer.phone
IS 'MASKED WITH FUNCTION anon.partial(phone,2,$$X-XXX-XX$$,2)';

SELECT anon.anonymize_column('customer','phone');

SELECT * FROM customer ;
```

Les numéros de téléphone apparaissent encore car ils ne sont pas masqués à l'utilisateur en cours. Il faut le déclarer pour les utilisateurs concernés :

```
CREATE ROLE soustraitant LOGIN ;
\password soustraitant

GRANT SELECT ON customer TO soustraitant ;
SECURITY LABEL FOR anon ON ROLE soustraitant  IS 'MASKED';
```

Ce nouvel utilisateur verra à chaque fois des noms différents (masquage dynamique), et des numéros de téléphone partiellement masqués :

```
\c sensible soustraitant
SELECT * FROM customer ;
```

```
 id  | firstname | lastname |    phone     |   birth    | postcode
-----+-----------+----------+--------------+------------+----------
 107 | Sarah     | Kovacek  | 06X-XXX-XX11 | 1965-10-10 | 90016
 258 | Luke      | Effertz  | ø            | 1951-09-25 | 90120
 341 | Don       | Turcotte | 34X-XXX-XX23 | 1926-06-01 | 04520
```

Pour consulter la configuration de masquage en place, utiliser une des vues fournies dans le schéma `anon` :

```
=# SELECT * FROM anon.pg_masks \gx

-[ RECORD 1 ]----+-------------------------------------------------------------
attrelid         | 41853
attnum           | 3
relnamespace     | public
relname          | customer
attname          | lastname
format_type      | text
col_description  | MASKED WITH FUNCTION anon.fake_last_name()
masking_function | anon.fake_last_name()
masking_value    |
priority         | 100
masking_filter   | anon.fake_last_name()
```

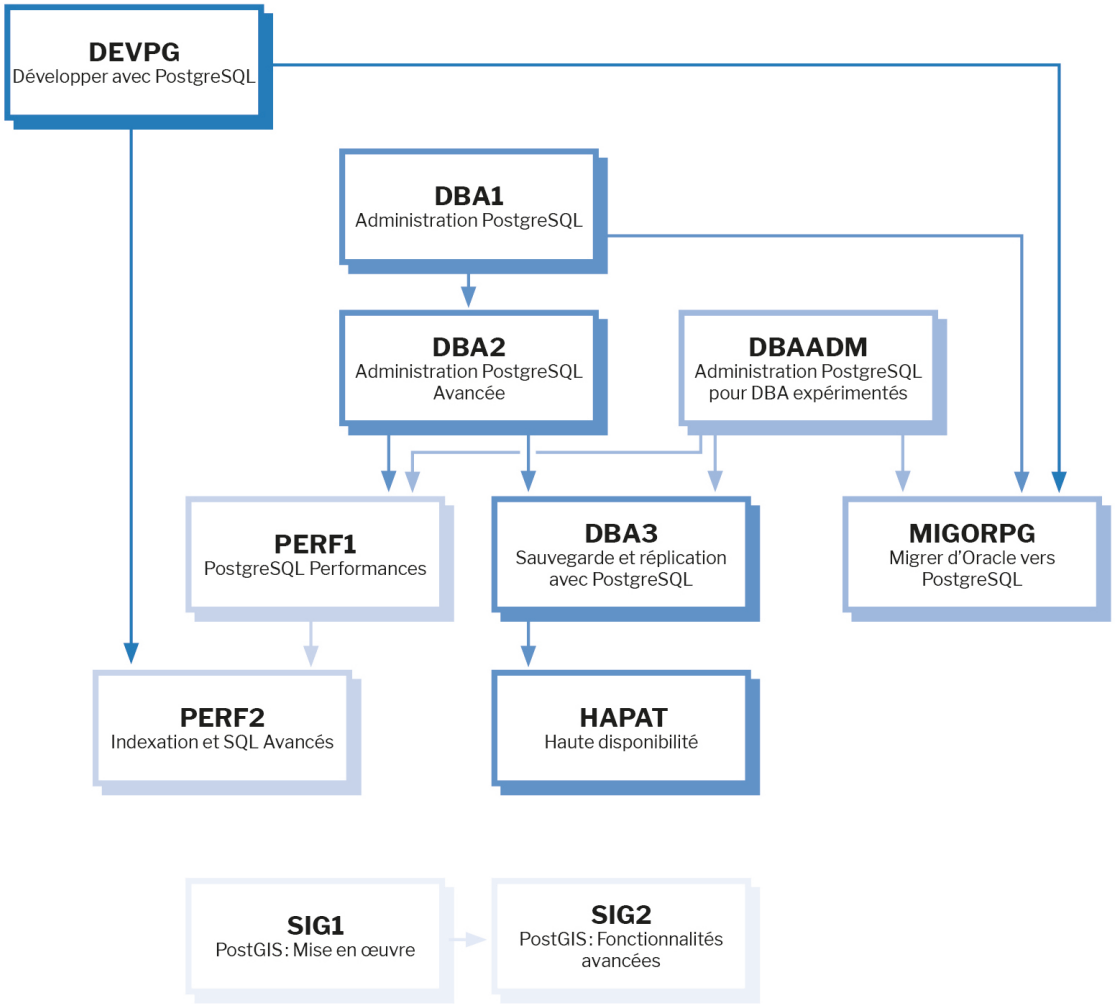
```
trusted_schema    | t
-[ RECORD 2 ]-----+-------------------------------------------------------
attrelid          | 41853
attnum            | 4
relnamespace      | public
relname           | customer
attname           | phone
format_type       | text
col_description   | MASKED WITH FUNCTION anon.partial(phone,2,$$X-XXX-XX$$,2)
masking_function  | anon.partial(phone,2,$$X-XXX-XX$$,2)
masking_value     |
priority          | 100
masking_filter    | anon.partial(phone,2,$$X-XXX-XX$$,2)
trusted_schema    | t
```

Les formations Dalibo

Retrouvez nos formations et le calendrier sur https://dali.bo/formation

Pour toute information ou question, n'hésitez pas à nous écrire sur contact@dalibo.com.

Cursus des formations



Retrouvez nos formations dans leur dernière version :

- DBA1 : Administration PostgreSQL
  https://dali.bo/dba1
- DBA2 : Administration PostgreSQL avancé
  https://dali.bo/dba2
- DBA3 : Sauvegarde et réplication avec PostgreSQL
  https://dali.bo/dba3
- DEVPG : Développer avec PostgreSQL
  https://dali.bo/devpg
- PERF1 : PostgreSQL Performances
  https://dali.bo/perf1
- PERF2 : Indexation et SQL avancés
  https://dali.bo/perf2
- MIGORPG : Migrer d'Oracle à PostgreSQL
  https://dali.bo/migorpg
- HAPAT : Haute disponibilité avec PostgreSQL
  https://dali.bo/hapat

Les livres blancs

- Migrer d'Oracle à PostgreSQL
  https://dali.bo/dlb01
- Industrialiser PostgreSQL
  https://dali.bo/dlb02
- Bonnes pratiques de modélisation avec PostgreSQL
  https://dali.bo/dlb04
- Bonnes pratiques de développement avec PostgreSQL
  https://dali.bo/dlb05

Téléchargement gratuit

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open source ou sous licence Creative Commons.