**eBook Gratuit**

# APPRENEZ postgresql

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#postgresql

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: postgresql

It is an unofficial and free postgresql ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official postgresql.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Démarrer avec postgresql

## Remarques

Cette section fournit une vue d'ensemble de ce que postgresql est et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans postgresql, et établir un lien avec les sujets connexes. La documentation de postgresql étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

## Versions

| Version | Date de sortie | Date de fin de vie |
|---|---|---|
| 9,6 | 2016-09-29 | 2021-09-01 |
| 9,5 | 2016-01-07 | 2021-01-01 |
| 9.4 | 2014-12-18 | 2019-12-01 |
| 9.3 | 2013-09-09 | 2018-09-01 |
| 9.2 | 2012-09-10 | 2017-09-01 |
| 9.1 | 2011-09-12 | 2016-09-01 |
| 9.0 | 2010-09-20 | 2015-09-01 |
| 8.4 | 2009-07-01 | 2014-07-01 |

## Examples

### Installation sur GNU + Linux

Sur la plupart des systèmes d'exploitation GNU + Linux, PostgreSQL peut facilement être installé à l'aide du gestionnaire de packages du système d'exploitation.

## Famille Red Hat

Vous trouverez des réponses à vos questions ici: https://yum.postgresql.org/repopackages.php

Téléchargez le référentiel sur la machine locale avec la commande

```
yum -y install https://download.postgresql.org/pub/repos/yum/X.X/redhat/rhel-7-x86_64/pgdg-
```

```
redhatXX-X.X-X.noarch.rpm
```

Afficher les packages disponibles:

```
yum list available | grep postgres*
```

Les paquets nécessaires sont: postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Ceux-ci sont installés avec la commande suivante: yum -y installe postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Une fois installé, vous devrez démarrer le service de base de données en tant que propriétaire du service (la valeur par défaut est postgres). Ceci est fait avec la commande pg_ctl.

```
sudo -su postgres
./usr/pgsql-X.X/bin/pg_ctl -D /var/lib/pgsql/X.X/data start
```

Pour accéder à la base de données en CLI, entrez `psql`

# Famille Debian

Sur Debian et les systèmes d'exploitation dérivés , tapez:

```
sudo apt-get install postgresql
```

Cela installera le package du serveur PostgreSQL, à la version par défaut proposée par les référentiels de packages du système d'exploitation.

Si la version installée par défaut n'est pas celle que vous souhaitez, vous pouvez utiliser le gestionnaire de packages pour rechercher des versions spécifiques pouvant être proposées simultanément.

Vous pouvez également utiliser le référentiel Yum fourni par le projet PostgreSQL (appelé PGDG ) pour obtenir une version différente. Cela peut autoriser les versions non encore proposées par les référentiels de packages du système d'exploitation.

## Comment installer PostgreSQL via MacPorts sur OSX

Pour installer PostgreSQL sur OSX, vous devez savoir quelles versions sont actuellement prises en charge.

Utilisez cette commande pour voir quelles versions sont disponibles.

```
sudo port list | grep "^postgresql[[:digit:]]\{2\}[[:space:]]"
```

Vous devriez obtenir une liste qui ressemble à ce qui suit:

```
postgresql80                    @8.0.26         databases/postgresql80
postgresql81                    @8.1.23         databases/postgresql81
postgresql82                    @8.2.23         databases/postgresql82
postgresql83                    @8.3.23         databases/postgresql83
postgresql84                    @8.4.22         databases/postgresql84
postgresql90                    @9.0.23         databases/postgresql90
postgresql91                    @9.1.22         databases/postgresql91
postgresql92                    @9.2.17         databases/postgresql92
postgresql93                    @9.3.13         databases/postgresql93
postgresql94                    @9.4.8          databases/postgresql94
postgresql95                    @9.5.3          databases/postgresql95
postgresql96                    @9.6beta2       databases/postgresql96
```

Dans cet exemple, la version la plus récente de PostgreSQL est prise en charge en 9.6, nous allons donc l'installer.

```
sudo port install postgresql96-server postgresql96
```

Vous verrez un journal d'installation comme ceci:

```
--->  Computing dependencies for postgresql96-server
--->  Dependencies to be installed: postgresql96
--->  Fetching archive for postgresql96
--->  Attempting to fetch postgresql96-9.6beta2_0.darwin_15.x86_64.tbz2 from
https://packages.macports.org/postgresql96
--->  Attempting to fetch postgresql96-9.6beta2_0.darwin_15.x86_64.tbz2.rmd160 from
https://packages.macports.org/postgresql96
--->  Installing postgresql96 @9.6beta2_0
--->  Activating postgresql96 @9.6beta2_0

To use the postgresql server, install the postgresql96-server port

--->  Cleaning postgresql96
--->  Fetching archive for postgresql96-server
--->  Attempting to fetch postgresql96-server-9.6beta2_0.darwin_15.x86_64.tbz2 from
https://packages.macports.org/postgresql96-server
--->  Attempting to fetch postgresql96-server-9.6beta2_0.darwin_15.x86_64.tbz2.rmd160 from
https://packages.macports.org/postgresql96-server
--->  Installing postgresql96-server @9.6beta2_0
--->  Activating postgresql96-server @9.6beta2_0

To create a database instance, after install do
 sudo mkdir -p /opt/local/var/db/postgresql96/defaultdb
 sudo chown postgres:postgres /opt/local/var/db/postgresql96/defaultdb
 sudo su postgres -c '/opt/local/lib/postgresql96/bin/initdb -D
/opt/local/var/db/postgresql96/defaultdb'

--->  Cleaning postgresql96-server
--->  Computing dependencies for postgresql96
--->  Cleaning postgresql96
--->  Updating database of binaries
--->  Scanning binaries for linking errors
--->  No broken files found.
```

Le journal fournit des instructions sur le reste des étapes d'installation, nous le faisons ensuite.

```
sudo mkdir -p /opt/local/var/db/postgresql96/defaultdb
```

```
sudo chown postgres:postgres /opt/local/var/db/postgresql96/defaultdb
sudo su postgres -c '/opt/local/lib/postgresql96/bin/initdb -D
/opt/local/var/db/postgresql96/defaultdb'
```

Maintenant, nous démarrons le serveur:

```
sudo port load -w postgresql96-server
```

Vérifiez que nous pouvons nous connecter au serveur:

```
su postgres -c psql
```

Vous verrez une invite de postgres:

```
psql (9.6.1)
Type "help" for help.

postgres=#
```

Ici, vous pouvez saisir une requête pour vérifier que le serveur est en cours d'exécution.

```
postgres=#SELECT setting FROM pg_settings WHERE name='data_directory';
```

Et voir la réponse:

```
               setting
-----------------------------------------
/opt/local/var/db/postgresql96/defaultdb
(1 row)
postgres=#
```

Tapez \ q pour quitter:

```
postgres=#\q
```

Et vous serez de retour à votre invite de shell.

Toutes nos félicitations! Vous avez maintenant une instance PostgreSQL en cours d'exécution sur OS / X.

## Postgres.app pour Mac OSX

Un outil extrêmement simple pour installer PostgreSQL sur un Mac est disponible en téléchargeant Postgres.app .
Vous pouvez modifier les préférences pour que PostgreSQL fonctionne en arrière-plan ou uniquement lorsque l'application est en cours d'exécution.

## Installer PostgreSQL sur Windows

Bien qu'il soit recommandé d'utiliser un système d'exploitation basé sur Unix (par exemple, Linux ou BSD) en tant que serveur de production, vous pouvez facilement installer PostgreSQL sur Windows (uniquement en tant que serveur de développement).

Téléchargez les fichiers binaires d'installation Windows à partir de EnterpriseDB: http://www.enterprisedb.com/products-services-training/pgdownload Il s'agit d'une société tierce créée par les principaux contributeurs au projet PostgreSQL qui ont optimisé les fichiers binaires pour Windows.

Sélectionnez la dernière version stable (non bêta) (9.5.3 au moment de l'écriture). Vous voudrez probablement le package Win x86-64, mais si vous utilisez une version 32 bits de Windows, ce qui est courant sur les anciens ordinateurs, sélectionnez plutôt Win x86-32.

Remarque: le passage entre les versions bêta et stable impliquera des tâches complexes telles que le vidage et la restauration. La mise à niveau au sein de la version bêta ou stable nécessite uniquement un redémarrage du service.

Vous pouvez vérifier si votre version de Windows est 32 ou 64 bits en allant dans Panneau de configuration -> Système et sécurité -> Système -> Type de système, qui indiquera "##-bit Operating System". C'est le chemin pour Windows 7, il peut être légèrement différent sur les autres versions de Windows.

Dans le programme d'installation, sélectionnez les packages que vous souhaitez utiliser. Par exemple:

- pgAdmin ( https://www.pgadmin.org ) est une interface graphique gratuite pour la gestion de votre base de données et je le recommande vivement. En 9.6, ce sera installé par défaut.
- PostGIS ( http://postgis.net ) fournit des fonctionnalités d'analyse géospatiale sur les coordonnées GPS, les distances, etc. très populaires parmi les développeurs SIG.
- Le package linguistique fournit les bibliothèques requises pour le langage procédural officiellement pris en charge PL / Python, PL / Perl et PL / Tcl.
- D'autres packages, tels que pgAgent, pgBouncer et Slony, sont utiles pour les serveurs de production de plus grande taille, uniquement vérifiés en fonction des besoins.

Tous ces packages optionnels peuvent être installés ultérieurement via "Application Stack Builder".

Remarque: Il existe également d'autres langues non officiellement prises en charge telles que PL / V8 , PL / Lua PL / Java disponibles.

Ouvrez pgAdmin et connectez-vous à votre serveur en double-cliquant sur son nom, par ex. "PostgreSQL 9.5 (localhost: 5432).

À partir de là, vous pouvez suivre des guides tels que l'excellent livre PostgreSQL: Up and Running, 2nd Edition ( http://shop.oreilly.com/product/0636920032144.do ).

**Facultatif: Type de démarrage du service manuel**

PostgreSQL s'exécute en arrière-plan comme un service légèrement différent de la plupart des

programmes. Ceci est courant pour les bases de données et les serveurs Web. Son type de démarrage par défaut est Automatique, ce qui signifie qu'il fonctionnera toujours sans aucune intervention de votre part.

Pourquoi voudriez-vous contrôler manuellement le service PostgreSQL? Si vous utilisez votre PC comme serveur de développement de temps en temps et que vous l'utilisez également pour jouer à des jeux vidéo par exemple, PostegreSQL peut ralentir un peu votre système pendant son fonctionnement.

Pourquoi ne voudriez-vous pas un contrôle manuel? Démarrer et arrêter le service peut être un problème si vous le faites souvent.

Si vous ne remarquez aucune différence de vitesse et que vous préférez éviter les problèmes, laissez son type de démarrage sur Automatique et ignorez le reste de ce guide. Autrement...

Allez dans Panneau de configuration -> Système et sécurité -> Outils d'administration.

Sélectionnez "Services" dans la liste, cliquez avec le bouton droit sur son icône, puis sélectionnez Envoyer vers -> Bureau pour créer une icône de bureau pour un accès plus pratique.

Fermez la fenêtre Outils d'administration, puis lancez Services à partir de l'icône du bureau que vous venez de créer.

Faites défiler jusqu'à ce que vous voyiez un service avec un nom comme postgresql-x ## - 9. # (ex. "Postgresql-x64-9.5").

Cliquez avec le bouton droit sur le service postgres, sélectionnez Propriétés -> Type de démarrage -> Manuel -> Appliquer -> OK. Vous pouvez le ramener à l'automatique tout aussi facilement.

Si vous voyez d'autres services liés à PostgreSQL dans la liste tels que "pgbouncer" ou "PostgreSQL Scheduling Agent - pgAgent", vous pouvez également modifier leur type de démarrage en Manuel car ils ne sont pas très utiles si PostgreSQL n'est pas en cours d'exécution. Bien que cela signifie plus de tracas chaque fois que vous commencez et arrêtez, c'est à vous de décider. Ils n'utilisent pas autant de ressources que PostgreSQL ™ lui-même et peuvent ne pas avoir d'impact significatif sur les performances de votre système.

Si le service est en cours d'exécution, son statut indique Commencé, sinon il ne fonctionne pas.

Pour le démarrer, cliquez avec le bouton droit et sélectionnez Démarrer. Une invite de chargement sera affichée et devrait disparaître tout de suite après. Si cela vous donne une erreur, essayez une seconde fois. Si cela ne fonctionne pas, il y a eu un problème avec l'installation, peut-être parce que vous avez modifié certains paramètres de Windows, la plupart des gens ne changent pas.

Pour arrêter les postgres, faites un clic droit sur le service et sélectionnez Arrêter.

Si vous rencontrez un problème lors de la tentative de connexion à votre base de données, vérifiez les services pour vous assurer de leur exécution.

Pour d'autres détails très spécifiques sur l'installation d'EDB PostgreSQL, par exemple la version runtime python dans le pack de langue officiel d'une version PostgreSQL spécifique, reportez-vous toujours au guide d'installation EBD officiel , modifiez la version en lien avec la version majeure de votre installateur.

## Installer postgresql avec brew sur Mac

Homebrew s'appelle " *le gestionnaire de paquets manquant pour macOS* ". Il peut être utilisé pour créer et installer des applications et des bibliothèques. Une fois installé , vous pouvez utiliser la commande `brew` pour installer PostgreSQL et ses dépendances comme suit:

```
brew update
brew install postgresql
```

Homebrew installe généralement la dernière version stable. Si vous en avez besoin d'un autre, alors la `brew search postgresql` les versions disponibles. Si vous avez besoin de PostgreSQL avec des options spécifiques, alors les `brew info postgresql` les options supportées. Si vous avez besoin d'une option de génération non prise en charge, vous devrez peut-être effectuer vous-même la construction, mais vous pourrez toujours utiliser Homebrew pour installer les dépendances communes.

**Démarrer le serveur:**

```
brew services start postgresql
```

**Ouvrez l'invite PostgreSQL**

```
psql
```

Si psql se plaint qu'il n'y a pas de base de données correspondante pour votre utilisateur, exécutez `createdb` .

## Installer PostgreSQL depuis Source sur Linux

Dépendances:

- Version de GNU> 3.80
- un compilateur ISO / ANSI (par exemple gcc)
- un extracteur comme tar ou gzip
- zlib-devel
- readline-deod pour libedit-devel

Sources: Lien vers la dernière source (9.6.3)

Maintenant, vous pouvez extraire les fichiers source:

```
tar -xzvf postgresql-9.6.3.tar.gz
```

Il y a un grand nombre d'options différentes pour la configuration de PostgreSQL:

Lien complet vers la procédure d'installation complète

Petite liste d'options disponibles:

- `--prefix=PATH` Chemin `--prefix=PATH` pour tous les fichiers
- `--exec-prefix=PATH` chemin pour le fichier de type architectural
- `--bindir=PATH` Chemin d'accès aux programmes exécutables
- `--sysconfdir=PATH` Chemin d'accès aux fichiers de configuration
- `--with-pgport=NUMBER` spécifie un port pour votre serveur
- `--with-perl` ajoute le support perl
- `--with-python` ajoute le support python
- `--with-openssl` ajoute le support openssl
- `--with-ldap` ajoute le support ldap
- `--with-blocksize=BLOCKSIZE` définit la taille de la page en Ko
  - `BLOCKSIZE` doit avoir une puissance de 2 et entre 1 et 32
- `--with-wal-segsize=SEGSIZE` définit la taille de la taille du segment WAL en Mo
  - `SEGSIZE` doit être une puissance de 2 entre 1 et 64

Allez dans le nouveau dossier créé et exécutez le script cofigure avec les options souhaitées:

```
./configure --exec=/usr/local/pgsql
```

Exécuter `make` pour créer les fichiers object

Exécuter `make install` pour installer PostgreSQL à partir des fichiers construits

Exécuter `make clean` pour ranger

Pour l'extension, changez le répertoire `cd contrib` , exécutez `make` et `make install`

Lire Démarrer avec postgresql en ligne: https://riptutorial.com/fr/postgresql/topic/885/demarrer-avec-postgresql

# Chapitre 2: Accéder aux données par programmation

## Examples

### Accéder à Postgresql depuis .NET en utilisant le fournisseur Npgsql

Npgsql est l'un des fournisseurs .NET les plus populaires pour Postgresql.

Une requête typique est effectuée en créant une commande, en liant des paramètres, puis en exécutant la commande. En C #:

```
var connString = "Host=myserv;Username=myuser;Password=mypass;Database=mydb";
using (var conn = new NpgsqlConnection(connString))
{
    var querystring = "INSERT INTO data (some_field) VALUES (@content)";

    conn.Open();
    // Create a new command with CommandText and Connection constructor
    using (var cmd = new NpgsqlCommand(querystring, conn))
    {
        // Add a parameter and set its type with the NpgsqlDbType enum
        var contentString = "Hello World!";
        cmd.Parameters.Add("@content", NpgsqlDbType.Text).Value = contentString;

        // Execute a query that returns no results
        cmd.ExecuteNonQuery();


        /* It is possible to reuse a command object and open connection instead of creating
new ones */

        // Create a new query and set its parameters
        int keyId = 101;
        cmd.CommandText = "SELECT primary_key, some_field FROM data WHERE primary_key =
@keyId";
        cmd.Parameters.Clear();
        cmd.Parameters.Add("@keyId", NpgsqlDbType.Integer).Value = keyId;

        // Execute the command and read through the rows one by one
        using (NpgsqlDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())   // Returns false for 0 rows, or after reading the last row
of the results
            {
                // read an integer value
                int primaryKey = reader.GetInt32(0);
                // or
                primaryKey = Convert.ToInt32(reader["primary_key"]);

                // read a text value
                string someFieldText = reader["some_field"].ToString();
            }
        }
```

```
    }
}    // the C# 'using' directive calls conn.Close() and conn.Dispose() for us
```

### Accéder à PostgreSQL avec le C-API

Le C-API est le moyen le plus puissant pour accéder à PostgreSQL et il est étonnamment confortable.

## Compilation et liaison

Pendant la compilation, vous devez ajouter le répertoire include de PostgreSQL, qui peut être trouvé avec `pg_config --includedir` , dans le chemin d'inclusion.
Vous devez `libpq.so` lien avec la bibliothèque partagée du client PostgreSQL ( `libpq.so` sous UNIX, `libpq.dll` sous Windows). Cette bibliothèque se trouve dans le répertoire de la bibliothèque PostgreSQL, qui peut être trouvé avec `pg_config --libdir` .

**Note:** Pour des raisons historiques, la bibliothèque s'appelle `libpq.so` et *non* `libpg.so` , qui est un piège populaire pour les débutants.

Étant donné que l'échantillon de code ci-dessous se trouve dans le fichier `coltype.c` , la compilation et la liaison se feraient avec

```
gcc -Wall -I "$(pg_config --includedir)" -L "$(pg_config --libdir)" -o coltype coltype.c -lpq
```

avec le compilateur GNU C (pensez à ajouter `-Wl,-rpath,"$(pg_config --libdir)"` pour ajouter le chemin de recherche de la bibliothèque) ou à

```
cl /MT /W4 /I <include directory> coltype.c <path to libpq.lib>
```

sous Windows avec Microsoft Visual C.

## Programme d'échantillon

```
/* necessary for all PostgreSQL client programs, should be first */
#include <libpq-fe.h>

#include <stdio.h>
#include <string.h>

#ifdef TRACE
#define TRACEFILE "trace.out"
#endif

int main(int argc, char **argv) {
#ifdef TRACE
    FILE *trc;
#endif
    PGconn *conn;
```

```
    PGresult *res;
    int rowcount, colcount, i, j, firstcol;
    /* parameter type should be guessed by PostgreSQL */
    const Oid paramTypes[1] = { 0 };
    /* parameter value */
    const char * const paramValues[1] = { "pg_database" };

    /*
     * Using an empty connectstring will use default values for everything.
     * If set, the environment variables PGHOST, PGDATABASE, PGPORT and
     * PGUSER will be used.
     */
    conn = PQconnectdb("");

    /*
     * This can only happen if there is not enough memory
     * to allocate the PGconn structure.
     */
    if (conn == NULL)
    {
        fprintf(stderr, "Out of memory connecting to PostgreSQL.\n");
        return 1;
    }

    /* check if the connection attempt worked */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        /*
         * Even if the connection failed, the PGconn structure has been
         * allocated and must be freed.
         */
        PQfinish(conn);
        return 1;
    }

#ifdef TRACE
    if (NULL == (trc = fopen(TRACEFILE, "w")))
    {
        fprintf(stderr, "Error opening trace file \"%s\"!\n", TRACEFILE);
        PQfinish(conn);
        return 1;
    }

    /* tracing for client-server communication */
    PQtrace(conn, trc);
#endif

    /* this program expects the database to return data in UTF-8 */
    PQsetClientEncoding(conn, "UTF8");

    /* perform a query with parameters */
    res = PQexecParams(
        conn,
        "SELECT column_name, data_type "
            "FROM information_schema.columns "
            "WHERE table_name = $1",
        1,            /* one parameter */
        paramTypes,
        paramValues,
        NULL,        /* parameter lengths are not required for strings */
```

```
        NULL,        /* all parameters are in text format */
        0            /* result shall be in text format */
    );

    /* out of memory or sever communication broken */
    if (NULL == res)
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        PQfinish(conn);
#ifdef TRACE
        fclose(trc);
#endif
        return 1;
    }

    /* SQL statement should return results */
    if (PGRES_TUPLES_OK != PQresultStatus(res))
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        PQfinish(conn);
#ifdef TRACE
        fclose(trc);
#endif
        return 1;
    }

    /* get count of result rows and columns */
    rowcount = PQntuples(res);
    colcount = PQnfields(res);

    /* print column headings */
    firstcol = 1;

    printf("Description of the table \"pg_database\"\n");

    for (j=0; j<colcount; ++j)
    {
        if (firstcol)
            firstcol = 0;
        else
            printf(": ");

        printf(PQfname(res, j));
    }

    printf("\n\n");

    /* loop through rosult rows */
    for (i=0; i<rowcount; ++i)
    {
        /* print all column data */
        firstcol = 1;

        for (j=0; j<colcount; ++j)
        {
            if (firstcol)
                firstcol = 0;
            else
                printf(": ");

            printf(PQgetvalue(res, i, j));
```

```
        }

        printf("\n");
    }

    /* this must be done after every statement to avoid memory leaks */
    PQclear(res);
    /* close the database connection and release memory */
    PQfinish(conn);
#ifdef TRACE
    fclose(trc);
#endif
    return 0;
}
```

## Accéder à PostgreSQL à partir de python en utilisant psycopg2

Vous pouvez trouver la description du pilote ici .

L'exemple rapide est:

```
import psycopg2

db_host = 'postgres.server.com'
db_port = '5432'
db_un = 'user'
db_pw = 'password'
db_name = 'testdb'

conn = psycopg2.connect("dbname={} host={} user={} password={}".format(
                        db_name,  db_host, db_un, db_pw),
                        cursor_factory=RealDictCursor)
cur = conn.cursor()
sql = 'select * from testtable where id > %s and id < %s'
args = (1, 4)
cur.execute(sql, args)

print(cur.fetchall())
```

Résultera:

```
[{'id': 2, 'fruit': 'apple'}, {'id': 3, 'fruit': 'orange'}]
```

## Accéder à PostgreSQL à partir de PHP en utilisant Pomm2

Sur les épaules des pilotes de bas niveau, il y a pomm . Il propose une approche modulaire, des convertisseurs de données, un support d'écoute / notification, un inspecteur de bases de données et bien plus encore.

En supposant que Pomm a été installé en utilisant composer, voici un exemple complet:

```
<?php
use PommProject\Foundation\Pomm;
$loader = require __DIR__ . '/vendor/autoload.php';
```

```
$pomm = new Pomm(['my_db' => ['dsn' => 'pgsql://user:pass@host:5432/db_name']]);

// TABLE comment (
// comment_id uuid PK, created_at timestamptz NN,
// is_moderated bool NN default false,
// content text NN CHECK (content !~ '^\s+$'), author_email text NN)
$sql = <<<SQL
SELECT
  comment_id,
  created_at,
  is_moderated,
  content,
  author_email
FROM comment
  INNER JOIN author USING (author_email)
WHERE
  age(now(), created_at) < $*::interval
ORDER BY created_at ASC
SQL;

// the argument will be converted as it is cast in the query above
$comments = $pomm['my_db']
    ->getQueryManager()
    ->query($sql, [DateInterval::createFromDateString('1 day')]);

if ($comments->isEmpty()) {
    printf("There are no new comments since yesterday.");
} else {
    foreach ($comments as $comment) {
        printf(
                "%s has posted at %s. %s\n",
                $comment['author_email'],
                $comment['created_at']->format("Y-m-d H:i:s"),
                $comment['is_moderated'] ? '[OK]' : '');
    }
}
```

Le module de gestionnaire de requêtes de Pomm échappe aux arguments de requête pour empêcher l'injection SQL. Lorsque les arguments sont convertis, ils sont également convertis d'une représentation PHP en valeurs Postgres valides. Le résultat est un itérateur, il utilise un curseur en interne. Chaque ligne est convertie à la volée, booléenne en booléen, horodatage à \ DateTime etc.

Lire Accéder aux données par programmation en ligne:
https://riptutorial.com/fr/postgresql/topic/2014/acceder-aux-donnees-par-programmation

# Chapitre 3: Astuce Postgres

## Examples

### DATEADD alternative à Postgres

- `SELECT CURRENT_DATE + '1 day'::INTERVAL`
- `SELECT '1999-12-11'::TIMESTAMP + '19 days'::INTERVAL`
- `SELECT '1 month'::INTERVAL + '1 month 3 days'::INTERVAL`

### Valeurs séparées par une virgule d'une colonne

```
SELECT
    string_agg(<TABLE_NAME>.<COLUMN_NAME>, ',')
FROM
    <SCHEMA_NAME>.<TABLE_NAME> T
```

### Supprimer les enregistrements en double de la table postgres

```
DELETE
    FROM <SCHEMA_NAME>.<Table_NAME>
WHERE
    ctid NOT IN
        (
        SELECT
            MAX(ctid)
        FROM
            <SCHEMA_NAME>.<TABLE_NAME>
        GROUP BY
            <SCHEMA_NAME>.<TABLE_NAME>.*
        )
;
```

### Mettre à jour la requête avec la jointure entre deux tables, car Postresql ne prend pas en charge la requête de jointure dans la mise à jour.

```
    update <SCHEMA_NAME>.<TABLE_NAME_1> AS A
    SET <COLUMN_1> = True
    FROM <SCHEMA_NAME>.<TABLE_NAME_2> AS B
    WHERE
        A.<COLUMN_2> = B.<COLUMN_2> AND
        A.<COLUMN_3> = B.<COLUMN_3>
```

### Différence entre deux horodatages par mois et par an

Différence par mois entre deux dates (horodatage)

```
select
    (
```

```
        (DATE_PART('year', AgeonDate) - DATE_PART('year', tmpdate)) * 12
        +
        (DATE_PART('month', AgeonDate) - DATE_PART('month', tmpdate))
    )
from dbo."Table1"
```

Différence par année entre deux dates (horodatage)

```
select (DATE_PART('year', AgeonDate) - DATE_PART('year', tmpdate)) from dbo."Table1"
```

## Requête pour copier / déplacer / transférer des données de table d'une base de données vers une autre table de base de données avec le même schéma

Premier Execute

```
CREATE EXTENSION DBLINK;
```

alors

```
INSERT INTO
    <SCHEMA_NAME>.<TABLE_NAME_1>
SELECT *
FROM
    DBLINK(
    'HOST=<IP-ADDRESS> USER=<USERNAME> PASSWORD=<PASSWORD> DBNAME=<DATABASE>',
    'SELECT * FROM <SCHEMA_NAME>.<TABLE_NAME_2>')
    AS <TABLE_NAME>
    (
    <COLUMN_1> <DATATYPE_1>,
    <COLUMN_1> <DATATYPE_2>,
    <COLUMN_1> <DATATYPE_3>
    );
```

Lire Astuce Postgres en ligne: https://riptutorial.com/fr/postgresql/topic/7433/astuce-postgres

# Chapitre 4: Commentaires dans postgresql

## Introduction

***Le*** but principal de ***COMMMENT*** est de définir ou de modifier un commentaire sur un objet de base de données.

Seul un commentaire (chaîne) peut être donné sur n'importe quel objet de base de données. COMMENT nous aidera à savoir ce qui a été défini pour l'objet de base de données particulier.

La règle pour ***COMMENT ON ROLE*** est que vous devez être superutilisateur pour commenter un rôle de superutilisateur ou avoir le privilège ***CREATEROLE*** pour commenter des rôles non super-utilisateur. Bien sûr, un *super - utilisateur peut commenter n'importe quoi*

## Syntaxe

- COMMENTAIRE objet_base_données nom_objet EST 'Texte';

## Remarques

Syntaxe complète: http://www.postgresql.org/docs/current/static/sql-comment.html

## Examples

### COMMENTAIRE sur la table

COMMENTAIRES SUR LA TABLE nom_table EST 'ceci est la table des détails de l'étudiant';

### Supprimer le commentaire

COMMENTAIRE sur l'étudiant de TABLE IS NULL;

Le commentaire sera supprimé avec l'exécution de l'instruction ci-dessus.

Lire Commentaires dans postgresql en ligne:
https://riptutorial.com/fr/postgresql/topic/8191/commentaires-dans-postgresql

# Chapitre 5: Création de table

## Examples

### Création de table avec clé primaire

```
CREATE TABLE person (
    person_id BIGINT NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255),
    PRIMARY KEY (person_id)
);
```

Vous pouvez également placer la contrainte `PRIMARY KEY` directement dans la définition de colonne:

```
CREATE TABLE person (
    person_id BIGINT NOT NULL PRIMARY KEY,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255)
);
```

Il est recommandé d'utiliser des noms de minuscules pour la table et toutes les colonnes. Si vous utilisez des noms en majuscules tels que `Person` vous devrez insérer ce nom entre guillemets ( `"Person"` ) dans chaque requête, car PostgreSQL applique le pliage de casse.

### Afficher la définition de la table

Ouvrez l'outil de ligne de commande `psql` connecté à la base de données où se trouve votre table. Ensuite, tapez la commande suivante:

```
\d tablename
```

Pour obtenir un type d'informations étendu

```
\d+ tablename
```

Si vous avez oublié le nom de la table, tapez simplement \ d dans psql pour obtenir une liste de tables et de vues dans la base de données actuelle.

### Créer une table à partir de select

Disons que vous avez une table appelée personne:

```
CREATE TABLE person (
```

```
    person_id BIGINT NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    age INT NOT NULL,
    PRIMARY KEY (person_id)
);
```

Vous pouvez créer une nouvelle table de personnes de plus de 30 ans comme ceci:

```
CREATE TABLE people_over_30 AS SELECT * FROM person WHERE age > 30;
```

## Créer un tableau non identifié

Vous pouvez créer des tableaux non liés afin de rendre les tableaux beaucoup plus rapides. Table unlogged saute l' écriture `write-ahead` journal qui signifie qu'il est pas Glissières de sécurité et incapable de se répliquer.

```
CREATE UNLOGGED TABLE person (
    person_id BIGINT NOT NULL PRIMARY KEY,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255)
);
```

## Créez une table qui référence une autre table.

Dans cet exemple, User Table aura une colonne qui référence la table Agency.

```
CREATE TABLE agencies ( -- first create the agency table
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL
)

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  agency_id NOT NULL INTEGER REFERENCES agencies(id) DEFERRABLE INITIALLY DEFERRED -- this is
going to references your agency table.
)
```

Lire Création de table en ligne: https://riptutorial.com/fr/postgresql/topic/2430/creation-de-table

# Chapitre 6: Dates, horodatages et intervalles

## Examples

### Lancer un horodatage ou un intervalle sur une chaîne

Vous pouvez convertir une valeur d' `timestamp` ou d' `interval` en chaîne avec la fonction `to_char()` :

```
SELECT to_char('2016-08-12 16:40:32'::timestamp, 'DD Mon YYYY HH:MI:SSPM');
```

Cette déclaration produira la chaîne "12 Aug 2016 16:40:32". La chaîne de formatage peut être modifiée de différentes manières. La liste complète des modèles de modèles peut être trouvée ici .

Notez que vous pouvez également insérer du texte brut dans la chaîne de formatage et que vous pouvez utiliser les modèles de modèle dans n'importe quel ordre:

```
SELECT to_char('2016-08-12 16:40:32'::timestamp,
              '"Today is "FMDay", the "DDth" day of the month of "FMMonth" of "YYYY');
```

Cela produira la chaîne "Aujourd'hui, c'est samedi, le 12e jour du mois d'août 2016". Vous devez cependant garder à l'esprit que tous les modèles de modèles - même ceux à une lettre comme "I", "D", "W" - sont convertis, à moins que le texte brut ne soit entre guillemets. Par mesure de sécurité, vous devez mettre tout le texte brut entre guillemets, comme ci-dessus.

Vous pouvez localiser la chaîne dans la langue de votre choix (noms des jours et des mois) en utilisant le modificateur TM (mode de traduction). Cette option utilise le paramètre de localisation du serveur exécutant PostgreSQL ou le client qui s'y connecte.

```
SELECT to_char('2016-08-12 16:40:32'::timestamp, 'TMDay, DD" de "TMMonth" del año "YYYY');
```

Avec un paramètre de langue espagnol, ceci produit "Sabado, 12 de Agosto del Año 2016".

### SÉLECTIONNEZ le dernier jour du mois

Vous pouvez sélectionner le dernier jour du mois.

```
SELECT (date_trunc('MONTH', ('201608'||'01')::date) + INTERVAL '1 MONTH - 1 day')::DATE;
```

`201608` est remplaçable par une variable.

### Compter le nombre d'enregistrements par semaine

SELECT date_trunc ('semaine', <>) AS "Semaine", comptez (*) FROM <> GROUP BY 1 ORDER BY 1;

Lire Dates, horodatages et intervalles en ligne:

https://riptutorial.com/fr/postgresql/topic/4227/dates--horodatages-et-intervalles

# Chapitre 7: Déclencheurs d'événement

## Introduction

Les déclencheurs d'événement seront déclenchés chaque fois qu'un événement associé à ces événements se produira dans la base de données.

## Remarques

Veuillez utiliser le lien ci-dessous pour un aperçu complet des déclencheurs d'événements dans PostgreSQL

https://www.postgresql.org/docs/9.3/static/event-trigger-definition.html

## Examples

### Journalisation des événements de démarrage de la commande DDL

Type d'événement-

- `DDL_COMMAND_START`
- `DDL_COMMAND_END`
- Le SQL_DROP

Ceci est un exemple de création d'un déclencheur d'événement et de journalisation des événements `DDL_COMMAND_START` .

```
CREATE TABLE TAB_EVENT_LOGS(
  DATE_TIME TIMESTAMP,
  EVENT_NAME TEXT,
  REMARKS TEXT
);

CREATE OR REPLACE FUNCTION FN_LOG_EVENT()
  RETURNS EVENT_TRIGGER
  LANGUAGE SQL
  AS
  $main$
    INSERT INTO TAB_EVENT_LOGS(DATE_TIME,EVENT_NAME,REMARKS)
      VALUES(NOW(),TG_TAG,'Event Logging');
  $main$;

CREATE EVENT TRIGGER TRG_LOG_EVENT ON DDL_COMMAND_START
  EXECUTE PROCEDURE FN_LOG_EVENT();
```

Lire Déclencheurs d'événement en ligne:
https://riptutorial.com/fr/postgresql/topic/9255/declencheurs-d-evenement

# Chapitre 8: Exporter l'en-tête et les données de la table de base de données PostgreSQL dans un fichier CSV

## Introduction

À partir de l'outil de gestion Adminer, il a l'option d'exportation vers le fichier csv pour la base de données mysql. Mais n'est pas disponible pour la base de données postgresql. Ici, je vais montrer la commande pour exporter CSV pour la base de données postgresql.

## Examples

### Exporter la table PostgreSQL vers csv avec en-tête pour certaines colonnes

```
COPY products(is_public, title, discount) TO 'D:\csv_backup\products_db.csv' DELIMITER ',' CSV
HEADER;

COPY categories(name) TO 'D:\csv_backup\categories_db.csv' DELIMITER ',' CSV HEADER;
```

### Sauvegarde de table complète sur csv avec en-tête

```
COPY products TO 'D:\csv_backup\products_db.csv' DELIMITER ',' CSV HEADER;

COPY categories TO 'D:\csv_backup\categories_db.csv' DELIMITER ',' CSV HEADER;
```

### copier de la requête

```
copy (select oid,relname from pg_class limit 5) to stdout;
```

Lire Exporter l'en-tête et les données de la table de base de données PostgreSQL dans un fichier CSV en ligne: https://riptutorial.com/fr/postgresql/topic/8643/exporter-l-en-tete-et-les-donnees-de-la-table-de-base-de-donnees-postgresql-dans-un-fichier-csv

# Chapitre 9: Expressions de table communes (WITH)

## Examples

### Expressions de table communes dans les requêtes SELECT

Les expressions de table communes prennent en charge l'extraction de parties de requêtes plus importantes. Par exemple:

```
WITH sales AS (
  SELECT
    orders.ordered_at,
    orders.user_id,
    SUM(orders.amount) AS total
  FROM orders
  GROUP BY orders.ordered_at, orders.user_id
)
SELECT
  sales.ordered_at,
  sales.total,
  users.name
FROM sales
JOIN users USING (user_id)
```

### Traversée d'arbre utilisant WITH RECURSIVE

```
create table empl (
    name text primary key,
    boss text null
        references name
            on update cascade
            on delete cascade
        default null
);

insert into empl values ('Paul',null);
insert into empl values ('Luke','Paul');
insert into empl values ('Kate','Paul');
insert into empl values ('Marge','Kate');
insert into empl values ('Edith','Kate');
insert into empl values ('Pam','Kate');
insert into empl values ('Carol','Luke');
insert into empl values ('John','Luke');
insert into empl values ('Jack','Carol');
insert into empl values ('Alex','Carol');

with recursive t(level,path,boss,name) as (
        select 0,name,boss,name from empl where boss is null
    union
        select
            level + 1,
```

```
            path || ' > ' || empl.name,
            empl.boss,
            empl.name
        from
            empl join t
                on empl.boss = t.name
) select * from t order by path;
```

Lire Expressions de table communes (WITH) en ligne:
https://riptutorial.com/fr/postgresql/topic/1973/expressions-de-table-communes--with-

# Chapitre 10: EXTENSION dblink et postgres_fdw

## Syntaxe

- dblink ('dbname = nom_db_distance port = hôte PortOfDB = HostOfDB user = nom d'utilisateurDB mot de passe = passwordDB', 'MY QUESRY')
- dbname = nom de la base de données
- port = Port de la base de données
- host = hôte de la base de données
- user = nom d'utilisateur de la base de données
- password = mot de passe de la base de données ',
- MY QUESRY = cela peut être n'importe quelle opération que je veux faire SELECT, INSERT, ...

## Examples

### Dblink d'extension

dblink EXTENSION est une technique permettant de connecter une autre base de données et de faire en sorte que cette base de données fonctionne de la manière suivante:

1-Créer une extension dblink:

```
CREATE EXTENSION dblink;
```

2-Faites votre opération:

Par exemple Sélectionnez un attribut d'une autre table dans une autre base de données:

```
SELECT * FROM
dblink ('dbname = bd_distance port = 5432 host = 10.6.6.6 user = username
password = passw@rd', 'SELECT id, code FROM schema.table')
AS newTable(id INTEGER, code character varying);
```

### Extention FDW

FDW est une implication de dblink, il est plus utile de l’utiliser:

1-Créer une extension:

```
CREATE EXTENSION postgres_fdw;
```

2-Create SERVER:

```
CREATE SERVER name_srv FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'hostname',
dbname 'bd_name', port '5432');
```

3-Créer un mappage utilisateur pour le serveur postgres

```
CREATE USER MAPPING FOR postgres SERVER name_srv OPTIONS(user 'postgres', password
'password');
```

4-Créer une table étrangère:

```
CREATE FOREIGN TABLE table_foreign (id INTEGER, code character varying)
SERVER name_srv OPTIONS(schema_name 'schema', table_name 'table');
```

5-utiliser cette table étrangère comme dans votre base de données:

```
SELECT * FROM table_foreign;
```

## Wrapper de données étrangères

Pour accéder au schéma complet du serveur db au lieu d'une table unique. Suivez les étapes ci-dessous:

1. Créer une extension:

```
CREATE EXTENSION postgres_fdw;
```

2. Créer SERVER:

```
CREATE SERVER server_name FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'host_ip',
dbname 'db_name', port 'port_number');
```

3. Créer une cartographie utilisateur:

```
CREATE USER MAPPING FOR CURRENT_USER
SERVER server_name
OPTIONS (user 'user_name', password 'password');
```

4. Créez un nouveau schéma pour accéder au schéma de la base de données du serveur:

```
CREATE SCHEMA schema_name;
```

5. Importer le schéma du serveur:

```
IMPORT FOREIGN SCHEMA schema_name_to_import_from_remote_db
```

```
    FROM SERVER server_name
    INTO schema_name;
```

6. Accédez à n'importe quelle table du schéma de serveur:

```
SELECT * FROM schema_name.table_name;
```

Cela peut être utilisé pour accéder à plusieurs schémas de la base de données distante.

Lire EXTENSION dblink et postgres_fdw en ligne:
https://riptutorial.com/fr/postgresql/topic/6970/extension-dblink-et-postgres-fdw

# Chapitre 11: Fonctions cryptographiques postgres

## Introduction

Dans Postgres, les fonctions cryptographiques peuvent être déverrouillées à l'aide du module pgcrypto. CREATE EXTENSION pgcrypto;

## Examples

### digérer

`DIGEST()` génèrent un hachage binaire des données données. Cela **peut** être utilisé pour créer un hachage aléatoire.

Utilisation: `digest(data text, type text) returns bytea`

Ou: `digest(data bytea, type text) returns bytea`

Exemples:

- `SELECT DIGEST('1', 'sha1')`
- `SELECT DIGEST(CONCAT(CAST(current_timestamp AS TEXT), RANDOM()::TEXT), 'sha1')`

Lire Fonctions cryptographiques postgres en ligne:
https://riptutorial.com/fr/postgresql/topic/9230/fonctions-cryptographiques-postgres

# Chapitre 12: Fonctions d'agrégat

## Examples

### Statistiques simples: min (), max (), avg ()

Afin de déterminer des statistiques simples d'une valeur dans une colonne d'une table, vous pouvez utiliser une fonction d'agrégat.

Si votre table `individuals` est:

| prénom | Âge |
|---|---|
| Allie | 17 |
| Amanda | 14 |
| Alana | 20 |

Vous pouvez écrire cette déclaration pour obtenir la valeur minimale, maximale et moyenne:

```
SELECT min(age), max(age), avg(age)
FROM individuals;
```

Résultat:

| min | max | avg |
|---|---|---|
| 14 | 20 | 17 |

### string_agg (expression, délimiteur)

Vous pouvez concaténer des chaînes séparées par un délimiteur en utilisant la fonction `string_agg()` .

Si votre table `individuals` est:

| prénom | Âge | Pays |
|---|---|---|
| Allie | 15 | Etats-Unis |
| Amanda | 14 | Etats-Unis |
| Alana | 20 | Russie |

Vous pouvez écrire l' `SELECT ... GROUP BY` pour obtenir les noms de chaque pays:

```
SELECT string_agg(name, ', ') AS names, country
FROM individuals
GROUP BY country;
```

Notez que vous devez utiliser une clause `GROUP BY` car `string_agg()` est une fonction d'agrégat.

**Résultat:**

| des noms | pays |
|---|---|
| Allie, Amanda | Etats-Unis |
| Alana | Russie |

Plus la fonction d'agrégation PostgreSQL décrite ici

## regr_slope (Y, X): pente de l'équation linéaire d'ajustement par les moindres carrés déterminée par les paires (X, Y)

Pour illustrer l'utilisation de regr_slope (Y, X), je l'ai appliqué à un problème réel. En Java, si vous ne nettoyez pas correctement la mémoire, la corbeille peut rester bloquée et remplir la mémoire. Vous enregistrez des statistiques toutes les heures sur l'utilisation de la mémoire de différentes classes et les chargez dans une base de données postgres pour analyse.

Tous les candidats à la fuite de mémoire auront tendance à consommer plus de mémoire à mesure que le temps passe. Si vous tracez cette tendance, vous imaginez une ligne en haut et à gauche:

```
    ^
    |
s   |  Legend:
i   |  *  - data point
z   |  -- - trend
e   |
(   |
b   |                  *
y   |                      --
t   |                    --
e   |              * --     *
s   |            --
)   |       *--       *
    |     --     *
    |  -- *
   ------------------------------------->
                     time
```

Supposons que vous ayez une table contenant des données d'histogramme de vidage de tas (un mappage des classes sur la quantité de mémoire consommée):

```
CREATE TABLE heap_histogram (
    -- when the heap histogram was taken
    histwhen timestamp without time zone NOT NULL,
    -- the object type bytes are referring to
    -- ex: java.util.String
    class character varying NOT NULL,
    -- the size in bytes used by the above class
    bytes integer NOT NULL
);
```

Pour calculer la pente de chaque classe, nous regroupons par classe. La clause HAVING> 0 garantit que nous n'obtenons que des candidats avec un slop positif (une ligne allant vers le haut et vers la gauche). Nous trions par la pente en descendant pour obtenir les classes avec le plus fort taux de mémoire en haut.

```
-- epoch returns seconds
SELECT class, REGR_SLOPE(bytes,extract(epoch from histwhen)) as slope
    FROM public.heap_histogram
    GROUP BY class
    HAVING REGR_SLOPE(bytes,extract(epoch from histwhen)) > 0
    ORDER BY slope DESC ;
```

Sortie:

```
         class              |        slope
----------------------------+---------------------
 java.util.ArrayList        |     71.7993806279174
 java.util.HashMap          |     49.0324576155785
 java.lang.String           |     31.7770770326123
 joe.schmoe.BusinessObject  |     23.2036817108056
 java.lang.ThreadLocal      |     20.9013528767851
```

À la sortie, nous voyons que la consommation de mémoire de java.util.ArrayList augmente le plus rapidement à 71,799 octets par seconde et fait potentiellement partie de la fuite de mémoire.

Lire Fonctions d'agrégat en ligne: https://riptutorial.com/fr/postgresql/topic/4803/fonctions-d-agregat

# Chapitre 13: Fonctions de déclenchement et de déclenchement

## Introduction

Le déclencheur sera associé à la table ou à la vue spécifiée et exécutera la fonction spécifiée nom_fonction lorsque certains événements se produisent.

## Remarques

Veuillez utiliser le lien ci-dessous pour un aperçu complet de:

- **Déclencheurs** : https://www.postgresql.org/docs/current/static/sql-createtrigger.html
- **Fonctions de déclenchement** : https://www.postgresql.org/docs/current/static/plpgsql-trigger.html

## Examples

### Fonction de déclenchement PL / pgSQL de base

C'est une fonction de déclenchement simple.

```
CREATE OR REPLACE FUNCTION my_simple_trigger_function()
RETURNS trigger AS
$BODY$

BEGIN
    -- TG_TABLE_NAME :name of the table that caused the trigger invocation
IF (TG_TABLE_NAME = 'users') THEN

    --TG_OP : operation the trigger was fired
  IF (TG_OP = 'INSERT') THEN
    --NEW.id is holding the new database row value (in here id is the id column in users
table)
    --NEW will return null for DELETE operations
    INSERT INTO log_table (date_and_time, description) VALUES (now(), 'New user inserted. User
ID: '|| NEW.id);
    RETURN NEW;

  ELSIF (TG_OP = 'DELETE') THEN
    --OLD.id is holding the old database row value (in here id is the id column in users
table)
    --OLD will return null for INSERT operations
    INSERT INTO log_table (date_and_time, description) VALUES (now(), 'User deleted.. User ID:
' || OLD.id);
    RETURN OLD;

  END IF;

RETURN null;
```

```
END IF;

END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
```

Ajout de cette fonction de déclenchement à la table `users`

```
CREATE TRIGGER my_trigger
AFTER INSERT OR DELETE
ON users
FOR EACH ROW
EXECUTE PROCEDURE my_simple_trigger_function();
```

### Type de déclencheurs

## Un déclencheur peut être spécifié pour tirer:

- `BEFORE` l'opération ne soit tentée sur une ligne - insérer, mettre à jour ou supprimer;
- `AFTER` l'opération soit terminée - insérer, mettre à jour ou supprimer;
- `INSTEAD OF` l'opération en cas d'insertions, de mises à jour ou de suppressions dans une vue.

## Déclencheur marqué:

- `FOR EACH ROW` est appelé une fois pour chaque ligne que l'opération modifie;
- `FOR EACH STATEMENT` est appelé onde pour toute opération donnée.

## Se préparer à exécuter des exemples

```
CREATE TABLE company (
    id          SERIAL PRIMARY KEY NOT NULL,
    name        TEXT NOT NULL,
    created_at  TIMESTAMP,
    modified_at TIMESTAMP DEFAULT NOW()
)

CREATE TABLE log (
    id          SERIAL PRIMARY KEY NOT NULL,
    table_name  TEXT NOT NULL,
    table_id    TEXT NOT NULL,
    description TEXT NOT NULL,
    created_at  TIMESTAMP DEFAULT NOW()
)
```

## Déclencheur à simple insertion

## Étape 1: créez votre fonction

```
CREATE OR REPLACE FUNCTION add_created_at_function()
  RETURNS trigger AS $BODY$
BEGIN
  NEW.created_at := NOW();
  RETURN NEW;
END $BODY$
LANGUAGE plpgsql;
```

## Étape 2: créez votre déclencheur

```
CREATE TRIGGER add_created_at_trigger
BEFORE INSERT
ON company
FOR EACH ROW
EXECUTE PROCEDURE add_created_at_function();
```

## Étape 3: testez-le

```
INSERT INTO company (name) VALUES ('My company');
SELECT * FROM company;
```

# Déclencheur à usages multiples

## Étape 1: créez votre fonction

```
CREATE OR REPLACE FUNCTION add_log_function()
  RETURNS trigger AS $BODY$
DECLARE
  vDescription TEXT;
  vId INT;
  vReturn RECORD;
BEGIN
    vDescription := TG_TABLE_NAME || ' ';
     IF (TG_OP = 'INSERT') THEN
        vId := NEW.id;
        vDescription := vDescription || 'added. Id: ' || vId;
        vReturn := NEW;
    ELSIF (TG_OP = 'UPDATE') THEN
        vId := NEW.id;
        vDescription := vDescription || 'updated. Id: ' || vId;
        vReturn := NEW;
    ELSIF (TG_OP = 'DELETE') THEN
        vId := OLD.id;
        vDescription := vDescription || 'deleted. Id: ' || vId;
        vReturn := OLD;
    END IF;

    RAISE NOTICE 'TRIGER called on % - Log: %', TG_TABLE_NAME, vDescription;
```

```
    INSERT INTO log
        (table_name, table_id, description, created_at)
        VALUES
        (TG_TABLE_NAME, vId, vDescription, NOW());

    RETURN vReturn;
END $BODY$
  LANGUAGE plpgsql;
```

## Étape 2: créez votre déclencheur

```
CREATE TRIGGER add_log_trigger
AFTER INSERT OR UPDATE OR DELETE
ON company
FOR EACH ROW
EXECUTE PROCEDURE add_log_function();
```

## Étape 3: testez-le

```
INSERT INTO company (name) VALUES ('Company 1');
INSERT INTO company (name) VALUES ('Company 2');
INSERT INTO company (name) VALUES ('Company 3');
UPDATE company SET name='Company new 2' WHERE name='Company 2';
DELETE FROM company WHERE name='Company 1';
SELECT * FROM log;
```

Lire Fonctions de déclenchement et de déclenchement en ligne:
https://riptutorial.com/fr/postgresql/topic/6957/fonctions-de-declenchement-et-de-declenchement

# Chapitre 14: Fonctions de fenêtre

## Examples

### exemple générique

Préparation des données:

```
create table wf_example(i int, t text,ts timestamptz,b boolean);
insert into wf_example select 1,'a','1970.01.01',true;
insert into wf_example select 1,'a','1970.01.01',false;
insert into wf_example select 1,'b','1970.01.01',false;
insert into wf_example select 2,'b','1970.01.01',false;
insert into wf_example select 3,'b','1970.01.01',false;
insert into wf_example select 4,'b','1970.02.01',false;
insert into wf_example select 5,'b','1970.03.01',false;
insert into wf_example select 2,'c','1970.03.01',true;
```

Fonctionnement:

```
select *
  , dense_rank() over (order by i) dist_by_i
  , lag(t) over () prev_t
  , nth_value(i, 6) over () nth
  , count(true) over (partition by i) num_by_i
  , count(true) over () num_all
  , ntile(3) over() ntile
from wf_example
;
```

Résultat:

```
 i | t |           ts           | b | dist_by_i | prev_t | nth | num_by_i | num_all | ntile
---+---+------------------------+---+-----------+--------+-----+----------+---------+-------
 1 | a | 1970-01-01 00:00:00+01 | f |         1 |        |   3 |        3 |       8 |     1
 1 | a | 1970-01-01 00:00:00+01 | t |         1 | a      |   3 |        3 |       8 |     1
 1 | b | 1970-01-01 00:00:00+01 | f |         1 | a      |   3 |        3 |       8 |     1
 2 | c | 1970-03-01 00:00:00+01 | t |         2 | b      |   3 |        2 |       8 |     2
 2 | b | 1970-01-01 00:00:00+01 | f |         2 | c      |   3 |        2 |       8 |     2
 3 | b | 1970-01-01 00:00:00+01 | f |         3 | b      |   3 |        1 |       8 |     2
 4 | b | 1970-02-01 00:00:00+01 | f |         4 | b      |   3 |        1 |       8 |     3
 5 | b | 1970-03-01 00:00:00+01 | f |         5 | b      |   3 |        1 |       8 |     3
(8 rows)
```

Explication:

*dist_by_i* : `dense_rank() over (order by i)` est comme un row_number par valeurs distinctes. Peut être utilisé pour le nombre de valeurs distinctes de i ( `count(DISTINCT i)` wold not work). Utilisez simplement la valeur maximale.

*prev_t* : `lag(t) over ()` est une valeur précédente de t sur toute la fenêtre. attention, il est nul pour

la première ligne.

*nth* : `nth_value(i, 6) over ()` est la valeur de la colonne de la sixième rangée i sur toute la fenêtre

*num_by_i* : `count(true) over (partition by i)` est une quantité de lignes pour chaque valeur de i

*num_all* : `count(true) over ()` est une quantité de lignes sur une fenêtre entière

*ntile* : `ntile(3) over()` divise la fenêtre entière en 3 (autant que possible) égales en quantité

### valeurs de colonne vs dense_rank vs rang vs row_number

ici vous pouvez trouver les fonctions.

Avec la table wf_example créée dans l'exemple précédent, exécutez:

```
select i
  , dense_rank() over (order by i)
  , row_number() over ()
  , rank() over (order by i)
from wf_example
```

Le résultat est:

```
 i | dense_rank | row_number | rank
---+------------+------------+------
 1 |          1 |          1 |    1
 1 |          1 |          2 |    1
 1 |          1 |          3 |    1
 2 |          2 |          4 |    4
 2 |          2 |          5 |    4
 3 |          3 |          6 |    6
 4 |          4 |          7 |    7
 5 |          5 |          8 |    8
```

- Ordres *de DENSE_RANK* **valeurs** de **i** par apparence dans la fenêtre. `i=1` apparaît, donc la première ligne a dense_rank, la prochaine et la troisième valeur ne changent pas, donc c'est `dense_rank` indique *1* - la première valeur n'a pas changé. quatrième ligne `i=2` , il est la deuxième valeur de *i* rencontré, donc `dense_rank` montre 2, andso pour la ligne suivante. Ensuite, il rencontre la valeur `i=3` à la 6ème rangée, donc il montre 3. Même pour le reste deux valeurs de *i* . La dernière valeur de `dense_rank` est donc le nombre de valeurs distinctes de *i* .

- *row_number les* commandes **ROWS** telles qu'elles sont listées.

- *rank* Ne pas confondre avec `dense_rank` cette fonction ordonne **ROW NUMBER** des valeurs **i** . Donc, il commence avec trois, mais a la valeur suivante 4, ce qui signifie que `i=2` (nouvelle valeur) a été rencontré à la ligne 4. Même `i=3` été rencontré à la ligne 6. Etc ..

Lire Fonctions de fenêtre en ligne: https://riptutorial.com/fr/postgresql/topic/7421/fonctions-de-fenetre

# Chapitre 15: Gestion des rôles

## Syntaxe

- `CREATE ROLE name [ [ WITH ] option [ ... ] ]`
- `CREATE USER name [ [ WITH ] option [ ... ] ]`
- `where option can be: SUPERUSER | NOSUPERUSER | CREATEDB | NOCREATEDB | CREATEROLE | NOCREATEROLE | CREATEUSER | NOCREATEUSER | INHERIT | NOINHERIT | LOGIN | NOLOGIN | CONNECTION LIMIT connlimit | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password' | VALID UNTIL 'timestamp' | IN ROLE role_name [, ...] | IN GROUP role_name [, ...] | ROLE role_name [, ...] | ADMIN role_name [, ...] | USER role_name [, ...] | SYSID uid`

## Examples

### Créer un utilisateur avec un mot de passe

En règle générale, évitez d'utiliser le rôle de base de données par défaut (souvent `postgres` ) dans votre application. Vous devriez plutôt créer un utilisateur avec des niveaux de privilèges inférieurs. Ici nous en faisons un appelé `niceusername` et lui donner un mot de passe `very-strong-password`

```
CREATE ROLE niceusername with PASSWORD 'very-strong-password' LOGIN;
```

Le problème est que les requêtes saisies dans la console `psql` sont enregistrées dans un fichier d'historique `.psql_history` dans le répertoire `.psql_history` de l'utilisateur et peuvent également être consignées dans le journal du serveur de base de données PostgreSQL, exposant ainsi le mot de passe.

Pour éviter cela, utilisez la commande `\password` pour définir le mot de passe utilisateur. Si l'utilisateur qui émet la commande est un superutilisateur, le mot de passe actuel ne sera pas demandé. (Doit être un superutilisateur pour modifier les mots de passe des superutilisateurs)

```
CREATE ROLE niceusername with LOGIN;
\password niceusername
```

### Créer un rôle et une base de données correspondante

Pour prendre en charge une application donnée, vous créez souvent un nouveau rôle et une nouvelle base de données.

Les commandes shell à exécuter seraient les suivantes:

```
$ createuser -P blogger
Enter password for the new role: ********
Enter it again: ********

$ createdb -O blogger blogger
```

Cela suppose que `pg_hba.conf` a été correctement configuré, ce qui ressemble probablement à ceci:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    sameuser        all             localhost               md5
local   sameuser        all                                     md5
```

## Accorder et révoquer des privilèges.

Supposons que nous ayons trois utilisateurs:

1. L'administrateur de la base de données> admin
2. L'application avec un accès complet pour ses données> read_write
3. L'accès en lecture seule> read_only

```
--ACCESS DB
REVOKE CONNECT ON DATABASE nova FROM PUBLIC;
GRANT  CONNECT ON DATABASE nova  TO user;
```

Avec les requêtes ci-dessus, les utilisateurs non approuvés ne peuvent plus se connecter à la base de données.

```
--ACCESS SCHEMA
REVOKE ALL     ON SCHEMA public FROM PUBLIC;
GRANT  USAGE   ON SCHEMA public  TO user;
```

La série de requêtes suivante révoque tous les privilèges des utilisateurs non authentifiés et fournit un ensemble limité de privilèges pour l'utilisateur `read_write` .

```
--ACCESS TABLES
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM PUBLIC ;
GRANT SELECT                         ON ALL TABLES IN SCHEMA public TO read_only ;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO read_write ;
GRANT ALL                            ON ALL TABLES IN SCHEMA public TO admin ;

--ACCESS SEQUENCES
REVOKE ALL   ON ALL SEQUENCES IN SCHEMA public FROM PUBLIC;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO read_only; -- allows the use of CURRVAL
GRANT UPDATE ON ALL SEQUENCES IN SCHEMA public TO read_write; -- allows the use of NEXTVAL and
SETVAL
GRANT USAGE  ON ALL SEQUENCES IN SCHEMA public TO read_write; -- allows the use of CURRVAL and
NEXTVAL
GRANT ALL    ON ALL SEQUENCES IN SCHEMA public TO admin;
```

## Modifier le chemin de recherche par défaut de l'utilisateur

Avec les commandes ci-dessous, le chemin de recherche par défaut de l'utilisateur peut être défini.

1. Vérifiez le chemin de recherche avant de définir le schéma par défaut.

```
postgres=# \c postgres user1
You are now connected to database "postgres" as user "user1".
postgres=> show search_path;
  search_path
---------------
 "$user",public
(1 row)
```

2. Définissez `search_path` avec `alter user` commande `alter user` pour ajouter un nouveau schéma `my_schema`

```
postgres=> \c postgres postgres
You are now connected to database "postgres" as user "postgres".
postgres=# alter user user1 set search_path='my_schema, "$user", public';
ALTER ROLE
```

3. Vérifiez le résultat après l'exécution.

```
postgres=# \c postgres user1
Password for user user1:
You are now connected to database "postgres" as user "user1".
postgres=> show search_path;
 search_path
-------------
 my_schema, "$user", public
(1 row)
```

Alternative:

```
postgres=# set role user1;
postgres=# show search_path;
 search_path
-------------
 my_schema, "$user", public
(1 row)
```

## Accordez des privilèges d'accès aux objets créés ultérieurement.

Supposons que nous ayons `three users` :

1. L'administrateur de la base de données> `admin`
2. L'application avec un accès complet pour ses données> `read_write`
3. L'accès en lecture seule> `read_only`

Avec les requêtes ci-dessous, vous pouvez définir des privilèges d'accès sur les objets créés ultérieurement dans un schéma spécifié.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT                    ON TABLES TO
read_only;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT,INSERT,DELETE,UPDATE ON TABLES TO
read_write;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT ALL                       ON TABLES TO
admin;
```

Vous pouvez également définir des privilèges d'accès sur les objets créés ultérieurement par l'utilisateur spécifié.

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin GRANT SELECT  ON TABLES TO read_only;
```

## Créer un utilisateur en lecture seule

```
CREATE USER readonly WITH ENCRYPTED PASSWORD 'yourpassword';
GRANT CONNECT ON DATABASE <database_name> to readonly;

GRANT USAGE ON SCHEMA public to readonly;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
```

Lire Gestion des rôles en ligne: https://riptutorial.com/fr/postgresql/topic/1572/gestion-des-roles

# Chapitre 16: Haute disponibilité PostgreSQL

## Examples

### Réplication dans PostgreSQL

- **Configuration du serveur primaire**
  - **Exigences:**
    - Utilisateur de réplication pour les activités de réplication
    - Répertoire pour stocker les archives WAL
  - **Créer un utilisateur de réplication**

    ```
    createuser -U postgres replication -P -c 5 --replication
    ```

    ```
    + option -P will prompt you for new password
    + option -c is for maximum connections. 5 connections are enough for replication
    + -replication will grant replication privileges to the user
    ```

  - **Créer un répertoire d'archivage dans le répertoire de données**

    ```
    mkdir $PGDATA/archive
    ```

  - **Editez le fichier pg_hba.conf**

    Ce fichier d'authentification de base hôte contient le paramètre pour l'authentification automatique du client. Ajouter ci-dessous l'entrée:

    ```
    #hosttype    database_name     user_name       hostname/IP      method
     host        replication       replication     <slave-IP>/32    md5
    ```

  - **Editez le fichier postgresql.conf**

    C'est le fichier de configuration de PostgreSQL.

    ```
    wal_level = hot_standby
    ```

    Ce paramètre détermine le comportement du serveur esclave.

    ```
    `hot_standby` logs what is required to accept read only queries on slave server.

    `streaming` logs what is required to just apply the WAL's on slave.

    `archive` which logs what is required for archiving.
    ```

    ```
    archive_mode=on
    ```

    Ce paramètre permet d'envoyer des segments WAL à l'emplacement d'archivage à

l'aide du paramètre `archive_command` .

```
archive_command = 'test ! -f /path/to/archivedir/%f && cp %p /path/to/archivedir/%f'
```

En gros, ce que `archive_command` fait, c'est de copier les segments WAL dans le répertoire d'archivage.

`wal_senders = 5` Il s'agit du nombre maximal de processus émetteur WAL.

Maintenant, redémarrez le serveur principal.

- **Sauvegarde du serveur primaire sur le serveur esclave**

  Avant de modifier le serveur, arrêtez le serveur principal.

  Important: Ne redémarrez le service que lorsque toutes les étapes de configuration et de sauvegarde sont terminées. Vous devez ouvrir le serveur de secours dans un état où il est prêt à être un serveur de sauvegarde. Cela signifie que tous les paramètres de configuration doivent être en place et que les bases de données doivent être déjà synchronisées. Sinon, la réplication en continu ne démarrera pas »

- **Exécutez maintenant l'utilitaire pg_basebackup**

  `pg_basebackup` utilitaire `pg_basebackup` copie les données du répertoire de données du serveur principal vers le répertoire de données esclave.

  ```
  $ pg_basebackup -h <primary IP> -D /var/lib/postgresql/<version>/main -U replication -v -P
  --xlog-method=stream
  ```

```
    -D: This is tells pg_basebackup where to the initial backup

    -h: Specifies the system where to look for the primary server

    -xlog-method=stream: This will force the pg_basebackup to open another connection and
stream enough xlog while backup is running.
                         It also ensures that fresh backup can be started without failing back
to using an archive.
```

- **Configuration du serveur de secours**

  Pour configurer le serveur de secours, vous devez modifier postgresql.conf et créer un nouveau fichier de configuration nommé recovery.conf.

  ```
  hot_standby = on
  ```

  Ceci spécifie si vous êtes autorisé à exécuter des requêtes lors de la récupération

  - **Création du fichier recovery.conf**

    ```
    standby_mode = on
    ```

    Définissez la chaîne de connexion sur le serveur principal. Remplacez par l'adresse IP externe du serveur principal. Remplacez par le mot de passe de l'utilisateur nommé replication

`primary_conninfo = 'host = port = 5432 user = mot de passe de réplication ='

(Facultatif) Définissez l'emplacement du fichier de déclenchement:

```
trigger_file = '/tmp/postgresql.trigger.5432'
```

Le chemin d'accès au `trigger_file` que vous spécifiez correspond à l'emplacement où vous pouvez ajouter un fichier lorsque vous souhaitez que le système bascule sur le serveur de secours. La présence du fichier "déclenche" le basculement. Vous pouvez également utiliser la commande pg_ctl promotion pour déclencher le basculement.

- **Démarrer le serveur de secours**

  Vous avez maintenant tout en place et vous êtes prêt à ouvrir le serveur de secours

**Attribution**

Cet article est en grande partie dérivé et attribué à la procédure de configuration de PostgreSQL pour la haute disponibilité et la réplication avec redondance d'UC , avec des modifications mineures du formatage et des exemples, ainsi que du texte supprimé. La source a été publiée sous la licence Creative Commons Public License 3.0 , qui est conservée ici.

Lire Haute disponibilité PostgreSQL en ligne: https://riptutorial.com/fr/postgresql/topic/5478/haute-disponibilite-postgresql

# Chapitre 17: Héritage

## Remarques

Une explication de la raison pour laquelle vous souhaitez utiliser l'héritage dans PostgreSQL est disponible ici: http://stackoverflow.com/a/3075248/653378

## Examples

### Créer des tables enfants

```
CREATE TABLE users (username text, email text);
CREATE TABLE simple_users () INHERITS (users);
CREATE TABLE users_with_password (password text) INHERITS (users);
```

Nos trois tableaux ressemblent à ceci:

## utilisateurs

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |

## simple_users

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |

## users_with_password

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |
| mot de passe | texte |

### Modification de tables

Créons deux tables simples:

```
CREATE TABLE users (username text, email text);
CREATE TABLE simple_users () INHERITS (users);
```

# Ajouter des colonnes

```
ALTER TABLE simple_users ADD COLUMN password text;
```

## simple_users

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |
| mot de passe | texte |

L'ajout de la même colonne à la table parente va fusionner la définition des deux colonnes:

```
ALTER TABLE users ADD COLUMN password text;
```

AVIS: fusion de la définition de la colonne "mot de passe" pour l'enfant "simple_users"

# Supprimer des colonnes

En utilisant nos tables modifiées:

```
ALTER TABLE users DROP COLUMN password;
```

## utilisateurs

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |

## simple_users

| Colonne | Type |
|---|---|
| Nom d'utilisateur | texte |
| email | texte |
| mot de passe | texte |

Depuis que nous avons ajouté la colonne à `simple_users` , PostgreSQL `simple_users` s'assure que cette colonne n'est pas supprimée.

Maintenant, si nous avions une autre table enfant, sa colonne de `password` aurait bien sûr été supprimée.

Lire Héritage en ligne: https://riptutorial.com/fr/postgresql/topic/5429/heritage

# Chapitre 18: INSÉRER

## Examples

### INSERT de base

Disons que nous avons un tableau simple appelé personne:

```
CREATE TABLE person (
    person_id BIGINT,
    name VARCHAR(255).
    age INT,
    city VARCHAR(255)
);
```

L'insertion la plus élémentaire consiste à insérer toutes les valeurs dans la table:

```
INSERT INTO person VALUES (1, 'john doe', 25, 'new york');
```

Si vous souhaitez insérer uniquement des colonnes spécifiques, vous devez indiquer explicitement quelles colonnes:

```
INSERT INTO person (name, age) VALUES ('john doe', 25);
```

Notez que si des contraintes existent sur la table, telles que NOT NULL, vous devrez inclure ces colonnes dans les deux cas.

### Insertion de plusieurs lignes

Vous pouvez insérer plusieurs lignes dans la base de données en même temps:

```
INSERT INTO person (name, age) VALUES
  ('john doe', 25),
  ('jane doe', 20);
```

### Insérer de select

Vous pouvez insérer des données dans une table à la suite d'une instruction select:

```
INSERT INTO person SELECT * FROM tmp_person WHERE age < 30;
```

Notez que la projection de la sélection doit correspondre aux colonnes requises pour l'insertion. Dans ce cas, la table `tmp_person` a les mêmes colonnes que `person` .

### Insérer des données en utilisant COPY

COPY est le mécanisme d'insertion en bloc de PostgreSQL. C'est un moyen pratique de transférer des données entre des fichiers et des tables, mais il est également beaucoup plus rapide que `INSERT` lorsque vous ajoutez plus de quelques milliers de lignes à la fois.

Commençons par créer un fichier de données exemple.

```
cat > samplet_data.csv

1,Yogesh
2,Raunak
3,Varun
4,Kamal
5,Hari
6,Amit
```

Et nous avons besoin d'un tableau à deux colonnes dans lequel ces données peuvent être importées.

```
CREATE TABLE copy_test(id int, name varchar(8));
```

Maintenant l'opération de copie proprement dite, cela créera six enregistrements dans la table.

```
COPY copy_test FROM '/path/to/file/sample_data.csv' DELIMITER ',';
```

Au lieu d'utiliser un fichier sur le disque, vous pouvez insérer des données depuis `stdin`

```
COPY copy_test FROM stdin DELIMITER ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 7,Amol
>> 8,Amar
>> \.
Time: 85254.306 ms

SELECT * FROM copy_test ;
 id |  name
----+--------
  1 | Yogesh
  3 | Varun
  5 | Hari
  7 | Amol
  2 | Raunak
  4 | Kamal
  6 | Amit
  8 | Amar
```

Vous pouvez également copier des données d'une table pour les classer comme suit:

```
COPY copy_test TO 'path/to/file/sample_data.csv'  DELIMITER ',';
```

Pour plus de détails sur COPY, vous pouvez vérifier ici

## INSERT données et valeurs RETURING

Si vous insérez des données dans une table avec une colonne d'incrémentation automatique et si vous souhaitez obtenir la valeur de la colonne d'incrémentation automatique.

Disons que vous avez une table appelée `my_table` :

```
CREATE TABLE my_table
(
id serial NOT NULL, -- serial data type is auto incrementing four-byte integer
name character varying,
contact_number integer,
CONSTRAINT my_table_pkey PRIMARY KEY (id)
);
```

Si vous voulez insérer des données dans `my_table` et obtenir l'identifiant de cette ligne:

```
INSERT INTO my_table(name, contact_number) VALUES ( 'USER', 8542621) RETURNING id;
```

La requête ci-dessus renvoie l'identifiant de la ligne où le nouvel enregistrement a été inséré.

## SELECT les données dans le fichier.

Vous pouvez copier le tableau et le coller dans un fichier.

```
postgres=# select * from my_table;
 c1 | c2 | c3
----+----+----
  1 |  1 |  1
  2 |  2 |  2
  3 |  3 |  3
  4 |  4 |  4
  5 |  5 |
(5 rows)

postgres=# copy my_table to '/home/postgres/my_table.txt' using delimiters '|' with null as
'null_string' csv header;
COPY 5
postgres=# \! cat my_table.txt
c1|c2|c3
1|1|1
2|2|2
3|3|3
4|4|4
5|5|null_string
```

## UPSERT - INSERT ... SUR LES CONFLITS, MISE À JOUR ...

depuis la version 9.5, postgres offre la fonctionnalité `UPSERT` avec l'instruction `INSERT` .

Disons que vous avez une table appelée my_table, créée dans plusieurs exemples précédents. Nous insérons une ligne, renvoyant la valeur PK de la ligne insérée:

```
b=# INSERT INTO my_table (name,contact_number) values ('one',333) RETURNING id;
 id
```

```
----
  2
(1 row)

INSERT 0 1
```

Maintenant, si nous essayons d'insérer une ligne avec une clé unique existante, cela générera une exception:

```
b=# INSERT INTO my_table values (2,'one',333);
ERROR:  duplicate key value violates unique constraint "my_table_pkey"
DETAIL:  Key (id)=(2) already exists.
```

La fonctionnalité Upsert offre la possibilité de l'insérer quand même, résolvant le conflit:

```
b=# INSERT INTO my_table values (2,'one',333) ON CONFLICT (id) DO UPDATE SET name =
my_table.name||' changed to: "two" at '||now() returning *;
 id |                       name                              | contact_number
----+-------------------------------------------------------------------------------------------
------------------+----------------
  2 | one changed to: "two" at 2016-11-23 08:32:17.105179+00 |            333
(1 row)

INSERT 0 1
```

Lire INSÉRER en ligne: https://riptutorial.com/fr/postgresql/topic/2561/inserer

# Chapitre 19: METTRE À JOUR

## Examples

### Mettre à jour toutes les lignes d'une table

Vous mettez à jour toutes les lignes de la table en fournissant simplement un `column_name = value` :

```
UPDATE person SET planet = 'Earth';
```

### Mettre à jour toutes les lignes répondant à une condition

```
UPDATE person SET state = 'NY' WHERE city = 'New York';
```

### Mise à jour de plusieurs colonnes dans la table

Vous pouvez mettre à jour plusieurs colonnes d'une table dans la même instruction en séparant les paires `col=val` par des virgules:

```
UPDATE person
   SET country = 'USA',
       state = 'NY'
WHERE city = 'New York';
```

### Mise à jour d'une table en fonction de l'adhésion à une autre table

Vous pouvez également mettre à jour des données dans une table en fonction des données d'une autre table:

```
UPDATE person
SET state_code = cities.state_code
FROM cities
WHERE cities.city = city;
```

Ici , nous joignons la `person city` colonne à la `cities city` colonne afin d'obtenir le code d'état de la ville. Ceci est ensuite utilisé pour mettre à jour la colonne `state_code` dans la table des `person` .

Lire METTRE À JOUR en ligne: https://riptutorial.com/fr/postgresql/topic/3136/mettre-a-jour

# Chapitre 20: Programmation avec PL / pgSQL

## Remarques

PL / pgSQL est le langage de programmation intégré de PostgreSQL pour l'écriture de fonctions qui s'exécutent dans la base de données elle-même, connue sous le nom de procédures stockées dans d'autres bases de données. Il étend SQL avec des boucles, des conditions et des types de retour. Bien que sa syntaxe puisse être étrange pour de nombreux développeurs, elle est beaucoup plus rapide que tout ce qui s'exécute sur le serveur d'application, car la surcharge de connexion à la base de données est éliminée, ce qui est particulièrement utile lorsque vous devez exécuter une requête. et soumettre une autre requête.

Bien que de nombreux autres langages procéduraux existent pour PostgreSQL, tels que PL / Python, PL / Perl et PLV8, PL / pgSQL est un point de départ commun pour les développeurs souhaitant écrire leur première fonction PostgreSQL car sa syntaxe est basée sur SQL. Il est également similaire à PL / SQL, le langage procédural natif d'Oracle, donc tout développeur familier avec PL / SQL trouvera le langage familier, et tout développeur qui a l'intention de développer des applications Oracle à l'avenir de PL / pgSQL à PL / SQL avec une relative facilité.

Il convient de souligner que d'autres langages procéduraux existent et que PL / pgSQL ne leur est pas nécessairement supérieur, y compris la vitesse, mais des exemples dans PL / pgSQL peuvent servir de point de référence commun pour d'autres langages utilisés pour écrire des fonctions PostgreSQL. PL / pgSQL a le plus de tutoriels et de livres de tous les PL et peut être un tremplin pour apprendre les langues avec moins de documentation.

Voici des liens vers des guides et des livres gratuits sur PL / pgSQL:

- La documentation officielle: https://www.postgresql.org/docs/current/static/plpgsql.html
- Tutoriel w3resource.com: http://www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php
- tutoriel postgres.cz: http://postgres.cz/wiki/PL/pgSQL_(en)
- PostgreSQL Server Programming, 2e édition: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-server-programming-second-edition
- Guide du développeur PostgreSQL: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-developers-guide

## Examples

### Fonction de base PL / pgSQL

Une simple fonction PL / pgSQL:

```
CREATE FUNCTION active_subscribers() RETURNS bigint AS $$
DECLARE
    -- variable for the following BEGIN ... END block
    subscribers integer;
BEGIN
```

```
    -- SELECT must always be used with INTO
    SELECT COUNT(user_id) INTO subscribers FROM users WHERE subscribed;
    -- function result
    RETURN subscribers;
EXCEPTION
    -- return NULL if table "users" does not exist
    WHEN undefined_table
    THEN RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Cela aurait pu être réalisé avec juste l'instruction SQL mais montre la structure de base d'une fonction.

Pour exécuter la fonction, faites:

```
select active_subscribers();
```

## Syntaxe PL / pgSQL

```
CREATE [OR REPLACE] FUNCTION functionName (someParameter 'parameterType')
RETURNS 'DATATYPE'
AS $_block_name_$
DECLARE
    --declare something
BEGIN
    --do something
    --return something
END;
$_block_name_$
LANGUAGE plpgsql;
```

## RETOURS Bloc

Options pour retourner dans une fonction PL / pgSQL:

- `Datatype` Liste de tous les types de données
- `Table(column_name column_type, ...)`
- `Setof 'Datatype' or 'table_column'`

## exceptions personnalisées

créer une exception personnalisée 'P2222':

```
create or replace function s164() returns void as
$$
begin
raise exception using message = 'S 164', detail = 'D 164', hint = 'H 164', errcode = 'P2222';
end;
$$ language plpgsql
;
```

créer une exception personnalisée n'attribuant pas d'erreur:

```
create or replace function s165() returns void as
$$
begin
raise exception '%','nothing specified';
end;
$$ language plpgsql
;
```

appel:

```
t=# do
$$
declare
 _t text;
begin
  perform s165();
  exception when SQLSTATE 'P0001' then raise info '%','state P0001 caught: '||SQLERRM;
  perform s164();

end;
$$
;
INFO:  state P0001 caught: nothing specified
ERROR:  S 164
DETAIL:  D 164
HINT:  H 164
CONTEXT:  SQL statement "SELECT s164()"
PL/pgSQL function inline_code_block line 7 at PERFORM
```

ici personnalisé P0001 traité, et P2222, non, abandonnant l'exécution.

Il est également très logique de conserver un tableau des exceptions, comme ici:
http://stackoverflow.com/a/2700312/5315974

Lire Programmation avec PL / pgSQL en ligne:
https://riptutorial.com/fr/postgresql/topic/5299/programmation-avec-pl---pgsql

# Chapitre 21: Rechercher longueur de chaîne / longueur de caractère

## Introduction

Pour obtenir la longueur des champs "caractère variable", "texte", utilisez char_length () ou character_length ().

## Examples

### Exemple pour obtenir la longueur d'un champ de caractère variable

Exemple 1, Requête: `SELECT char_length('ABCDE')`

Résultat:

    5

Exemple 2, requête: `SELECT character_length('ABCDE')`

Résultat:

    5

Lire Rechercher longueur de chaîne / longueur de caractère en ligne: https://riptutorial.com/fr/postgresql/topic/9695/rechercher-longueur-de-chaine---longueur-de-caractere

# Chapitre 22: Requêtes récursives

## Introduction

Il n'y a pas de requêtes récursives réelles!

## Examples

### Somme des nombres entiers

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Lien vers la documentation

Lire Requêtes récursives en ligne: https://riptutorial.com/fr/postgresql/topic/9025/requetes-recursives

# Chapitre 23: Sauvegarde et restauration

## Remarques

## Sauvegarde du système de fichiers au lieu d'utiliser `pg_dumpall` et `pg_dump`

Il est très important que si vous utilisez ceci, vous appelez la fonction `pg_start_backup()` avant et la fonction `pg_stop_backup()` après. Faire des sauvegardes du système de fichiers n'est pas sûr autrement; même un instantané ZFS ou FreeBSD du système de fichiers sauvegardé sans ces appels de fonctions placera la base de données en mode de récupération et risque de perdre des transactions.

J'éviterais de faire des sauvegardes de systèmes de fichiers plutôt que des sauvegardes classiques de Postgres, à la fois pour cette raison et parce que les fichiers de sauvegarde Postgres (en particulier dans le format personnalisé) sont extrêmement polyvalents pour prendre en charge des restaurations alternatives. Comme ils sont des fichiers uniques, ils sont aussi moins compliqués à gérer.

## Examples

### Sauvegarde d'une base de données

```
pg_dump -Fc -f DATABASE.pgsql DATABASE
```

`-Fc` sélectionne le "format de sauvegarde personnalisé" qui vous donne plus de puissance que le SQL brut; voir `pg_restore` pour plus de détails. Si vous voulez un fichier SQL complet, vous pouvez le faire à la place:

```
pg_dump -f DATABASE.sql DATABASE
```

ou même

```
pg_dump DATABASE > DATABASE.sql
```

### Restauration des sauvegardes

```
psql < backup.sql
```

Une alternative plus sûre utilise `-1` pour envelopper la restauration dans une transaction. Le `-f` spécifie le nom du fichier plutôt que d'utiliser la redirection du shell.

```
psql -1f backup.sql
```

Les fichiers au format personnalisé doivent être restaurés à l'aide de `pg_restore` avec l'option `-d` pour spécifier la base de données:

```
pg_restore -d DATABASE DATABASE.pgsql
```

Le format personnalisé peut également être reconverti en SQL:

```
pg_restore backup.pgsql > backup.sql
```

L'utilisation du format personnalisé est recommandée car vous pouvez choisir les éléments à restaurer et éventuellement activer le traitement parallèle.

Vous devrez peut-être faire un pg_dump suivi d'un pg_restore si vous effectuez une mise à niveau d'une version postgresql à une version plus récente.

### Sauvegarde de l'ensemble du cluster

```
$ pg_dumpall -f backup.sql
```

Cela fonctionne en arrière-plan en effectuant plusieurs connexions au serveur une fois pour chaque base de données et en exécutant `pg_dump` dessus.

Parfois, vous pourriez être tenté de définir cela comme une tâche cron, de sorte que vous voulez voir la date à laquelle la sauvegarde a été prise dans le cadre du nom de fichier:

```
$ postgres-backup-$(date +%Y-%m-%d).sql
```

Cependant, veuillez noter que cela pourrait produire des fichiers volumineux sur une base quotidienne. Postgresql dispose d'un mécanisme bien meilleur pour les sauvegardes régulières - archives WAL

La sortie de pg_dumpall est suffisante pour restaurer une instance Postgres à configuration identique, mais les fichiers de configuration dans `$PGDATA` ( `pg_hba.conf` et `postgresql.conf` ) ne font pas partie de la sauvegarde, vous devrez donc les sauvegarder séparément.

```
postgres=# SELECT pg_start_backup('my-backup');
postgres=# SELECT pg_stop_backup();
```

Pour effectuer une sauvegarde du système de fichiers, vous devez utiliser ces fonctions pour vous assurer que Postgres est dans un état cohérent pendant la préparation de la sauvegarde.

### Utilisation de la copie pour importer

# Pour copier des données d'un fichier CSV dans une table

```
COPY <tablename> FROM '<filename with path>';
```

**Pour insérer dans l' `user` de la table à partir d'un fichier nommé `user_data.csv` placé dans `/home/user/` :**

```
COPY user FROM '/home/user/user_data.csv';
```

# Pour copier des données d'un fichier séparé par des tuyaux vers une table

```
COPY user FROM '/home/user/user_data' WITH DELIMITER '|';
```

Note: En l'absence de l'option `with delimiter` , le délimiteur par défaut est la virgule ,

# Pour ignorer la ligne d'en-tête lors de l'importation du fichier

Utilisez l'option En-tête:

```
COPY user FROM '/home/user/user_data' WITH DELIMITER '|' HEADER;
```

Remarque: si des données sont citées, par défaut, les guillemets sont des guillemets doubles. Si les données sont citées en utilisant un autre caractère, utilisez l'option `QUOTE` ; Cependant, cette option est autorisée uniquement lors de l'utilisation du format CSV.

### Utiliser la copie pour exporter

# Copier le tableau en standard o / p

COPY <nomtable> TO STDOUT (DELIMITER '|');

Pour exporter un utilisateur de table vers la sortie standard:

COPY user TO STDOUT (DELIMITER '|');

# Pour copier une table dans un fichier

COPY utilisateur FROM '/ home / user / user_data' WITH DELIMITER '|';

# Pour copier la sortie de l'instruction SQL dans un fichier

COPY (instruction sql) TO '<nomfichier avec chemin>';

COPY (SELECT * FROM utilisateur WHERE nom_utilisateur LIKE 'A%') TO '/ home / utilisateur / user_data';

# Copier dans un fichier compressé

COPY user TO PROGRAM 'gzip> /home/user/user_data.gz';

Ici, le programme gzip est exécuté pour compresser les données de la table utilisateur.

## Utiliser psql pour exporter des données

Les données peuvent être exportées à l'aide de la commande copy ou en utilisant les options de ligne de commande de la commande psql.

**Pour exporter des données csv d'un utilisateur de table vers un fichier csv:**

```
psql -p \<port> -U \<username> -d \<database> -A -F<delimiter> -c\<sql to execute> \> \<output filename with path>

psql -p 5432 -U postgres -d test_database -A -F, -c "select * from user" > /home/user/user_data.csv
```

Ici, la combinaison de -A et -F fait l'affaire.

`-F` est de spécifier le délimiteur

```
-A or --no-align
```

Passe au mode de sortie non aligné. (Le mode de sortie par défaut est aligné autrement.)

Lire Sauvegarde et restauration en ligne:
https://riptutorial.com/fr/postgresql/topic/2291/sauvegarde-et-restauration

# Chapitre 24: Script de sauvegarde pour une base de données de production

## Syntaxe

- Le script vous permet de créer un répertoire de sauvegarde pour chaque exécution avec la syntaxe suivante: Nom du répertoire de sauvegarde de la base de données + date et heure d'exécution
- Exemple: prodDir22-11-2016-19h55
- Une fois créé, il crée deux fichiers de sauvegarde avec la syntaxe suivante: **Nom de la base de données + date et heure d'exécution**
- Exemple :
- dbprod22-11-2016-19h55.backup **(fichier de vidage)**
- dbprod22-11-2016-19h55.sql **(fichier SQL)**
- À la fin d'une exécution au **22-11-2016** @ **19h55** , nous obtenons:
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.backup
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.sql

## Paramètres

| paramètre | détails |
|---|---|
| save_db | Le répertoire de sauvegarde principal |
| dbProd | Le répertoire de sauvegarde secondaire |
| RENDEZ-VOUS AMOUREUX | La date de la sauvegarde au format spécifié |
| dbprod | Le nom de la base de données à enregistrer |
| /opt/postgres/9.0/bin/pg_dump | Le chemin du binaire pg_dump |
| -h | Spécifie le nom d'hôte de la machine sur laquelle le serveur est exécuté. Exemple: localhost |
| -p | Spécifie le port TCP ou l'extension de fichier de domaine Unix local sur lequel le serveur écoute les connexions. Exemple 5432 |
| -U | Nom d'utilisateur à connecter en tant que. |

## Remarques

1. S'il existe un outil de sauvegarde tel que HDPS ou Symantec Backup , il est nécessaire de

vider le répertoire de sauvegarde **avant chaque lancement** .

Pour éviter d'encombrer l'outil de sauvegarde car la sauvegarde des anciens fichiers est supposée être effectuée.

Pour activer cette fonctionnalité, veuillez ne pas commenter la ligne N ° 3.

```
rm -R / save_db / *
```

2. Dans le cas où le budget ne permet pas d'avoir un outil de sauvegarde, on peut toujours utiliser le planificateur de tâches ( commande cron ).

   La commande suivante permet d'éditer la table cron pour l'utilisateur actuel.

```
crontab -e
```

   Planifiez le lancement du script avec le calendrier à 23 heures.

```
0 23 * * * /saveProdDb.sh
```

# Examples

### saveProdDb.sh

En général, nous avons tendance à sauvegarder la base de données avec le client pgAdmin. Voici un script sh utilisé pour enregistrer la base de données (sous Linux) dans deux formats:

- **Fichier SQL** : pour un éventuel résumé de données sur n'importe quelle version de PostgreSQL.
- **Fichier de vidage** : pour une version supérieure à la version actuelle.

```
#!/bin/sh
cd /save_db
#rm -R /save_db/*
DATE=$(date +%d-%m-%Y-%Hh%M)
echo -e "Sauvegarde de la base du ${DATE}"
mkdir prodDir${DATE}
cd prodDir${DATE}

#dump file
/opt/postgres/9.0/bin/pg_dump -i -h localhost -p 5432 -U postgres -F c -b -w -v -f
"dbprod${DATE}.backup" dbprod

#SQL file
/opt/postgres/9.0/bin/pg_dump -i -h localhost -p 5432 -U postgres --format plain --verbose  -f
"dbprod${DATE}.sql" dbprod
```

Lire Script de sauvegarde pour une base de données de production en ligne:
https://riptutorial.com/fr/postgresql/topic/7974/script-de-sauvegarde-pour-une-base-de-donnees-

de-production

# Chapitre 25: Se connecter à PostgreSQL à partir de Java

## Introduction

L'API pour utiliser une base de données relationnelle à partir de Java est JDBC.

Cette API est implémentée par un pilote JDBC.

Pour l'utiliser, placez le fichier JAR avec le pilote sur le chemin de classe JAVA.

Cette documentation montre des exemples d'utilisation du pilote JDBC pour se connecter à une base de données.

## Remarques

### URL JDBC

L'URL JDBC peut prendre l'une des formes suivantes:

- `jdbc:postgresql://` *`host`* `[:` *`port`* `]/[` *`database`* `][` *`parameters`* `]`

  *`host`* défaut à `localhost` , *`port`* à 5432.
  Si *`host`* est une adresse IPv6, il doit être placé entre crochets.
  Le nom de base de données par défaut est identique au nom de l'utilisateur connecté.

  Pour implémenter le basculement, il est possible d'avoir plusieurs entrées *`host`* `[:` *`port`* `]` séparées par une virgule.
  Ils sont essayés à tour de rôle jusqu'à ce qu'une connexion réussisse.

- `jdbc:postgresql:` *`database`* `[` *`parameters`* `]`

- `jdbc:postgresql:/[` *`parameters`* `]`

  Ces formulaires sont destinés aux connexions à `localhost` .

*`parameters`* est une liste de paires *`key`* `[=` *`value`* `]` , dirigée par `?` et séparé par `&` . Si la *`value`* est manquante, il est supposé être `true` .

Un exemple:

```
jdbc:postgresql://localhost/test?user=fred&password=secret&ssl&sslfactory=org.postgresql.ssl.NonValidat
```

### Les références

- Spécification JDBC: http://download.oracle.com/otndocs/jcp/jdbc-4_2-mrel2-eval-spec/

- Pilote JDBC PostgreSQL: https://jdbc.postgresql.org/
- Documentation du pilote PostgreSQL JDBC: https://jdbc.postgresql.org/documentation/head/index.html

# Examples

## Connexion avec java.sql.DriverManager

C'est le moyen le plus simple de se connecter.

Tout d'abord, le pilote doit être *enregistré* avec `java.sql.DriverManager` afin qu'il sache quelle classe utiliser.
Cela se fait en chargeant la classe de pilote, généralement avec `java.lang.Class.forname( <driver class name> ) .`

```
/**
 * Connect to a PostgreSQL database.
 * @param url the JDBC URL to connect to; must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
 * @return a connection object for the established connection
 * @throws ClassNotFoundException if the driver class cannot be found on the Java class path
 * @throws java.sql.SQLException if the connection to the database fails
 */
private static java.sql.Connection connect(String url, String user, String password)
    throws ClassNotFoundException, java.sql.SQLException
{
    /*
     * Register the PostgreSQL JDBC driver.
     * This may throw a ClassNotFoundException.
     */
    Class.forName("org.postgresql.Driver");
    /*
     * Tell the driver manager to connect to the database specified with the URL.
     * This may throw an SQLException.
     */
    return java.sql.DriverManager.getConnection(url, user, password);
}
```

L'utilisateur et le mot de passe ne peuvent pas non plus être inclus dans l'URL JDBC, auquel cas vous n'avez pas à les spécifier dans l' `getConnection` méthode `getConnection` .

## Connexion avec java.sql.DriverManager et les propriétés

Au lieu de spécifier des paramètres de connexion tels que user et password (voir la liste complète ici ) dans l'URL ou un paramètre distinct, vous pouvez les empaqueter dans un objet `java.util.Properties` :

```
/**
 * Connect to a PostgreSQL database.
 * @param url the JDBC URL to connect to. Must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
```

```
 * @return a connection object for the established connection
 * @throws ClassNotFoundException if the driver class cannot be found on the Java class path
 * @throws java.sql.SQLException if the connection to the database fails
 */
private static java.sql.Connection connect(String url, String user, String password)
    throws ClassNotFoundException, java.sql.SQLException
{
    /*
     * Register the PostgreSQL JDBC driver.
     * This may throw a ClassNotFoundException.
     */
    Class.forName("org.postgresql.Driver");
    java.util.Properties props = new java.util.Properties();
    props.setProperty("user", user);
    props.setProperty("password", password);
    /* don't use server prepared statements */
    props.setProperty("prepareThreshold", "0");
    /*
     * Tell the driver manager to connect to the database specified with the URL.
     * This may throw an SQLException.
     */
    return java.sql.DriverManager.getConnection(url, props);
}
```

## Connexion avec javax.sql.DataSource à l'aide d'un pool de connexions

Il est courant d'utiliser `javax.sql.DataSource` avec JNDI dans des conteneurs de serveur d'applications, où vous enregistrez une source de données sous un nom et l'examinez chaque fois que vous avez besoin d'une connexion.

Ceci est un code qui montre comment les sources de données fonctionnent:

```
/**
 * Create a data source with connection pool for PostgreSQL connections
 * @param url the JDBC URL to connect to. Must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
 * @return a data source with the correct properties set
 */
private static javax.sql.DataSource createDataSource(String url, String user, String password)
{
    /* use a data source with connection pooling */
    org.postgresql.ds.PGPoolingDataSource ds = new org.postgresql.ds.PGPoolingDataSource();
    ds.setUrl(url);
    ds.setUser(user);
    ds.setPassword(password);
    /* the connection pool will have 10 to 20 connections */
    ds.setInitialConnections(10);
    ds.setMaxConnections(20);
    /* use SSL connections without checking server certificate */
    ds.setSslMode("require");
    ds.setSslfactory("org.postgresql.ssl.NonValidatingFactory");

    return ds;
}
```

Une fois que vous avez créé une source de données en appelant cette fonction, vous l'utilisez

comme ceci:

```
/* get a connection from the connection pool */
java.sql.Connection conn = ds.getConnection();

/* do some work */

/* hand the connection back to the pool - it will not be closed */
conn.close();
```

Lire Se connecter à PostgreSQL à partir de Java en ligne:
https://riptutorial.com/fr/postgresql/topic/9633/se-connecter-a-postgresql-a-partir-de-java

# Chapitre 26: SE FONDRE

## Introduction

Coalesce renvoie le premier argument null null à partir d'un ensemble d'arguments. Seul le premier argument non nul est return, tous les arguments suivants sont ignorés. La fonction évaluera null si tous les arguments sont nuls.

## Examples

### Un seul argument non nul

```
PGSQL> SELECT COALESCE(NULL, NULL, 'HELLO WORLD');

coalesce
--------
'HELLO WORLD'
```

### Plusieurs arguments non nuls

PGSQL> SELECT COALESCE (NULL, NULL, 'first non null', null, null, 'second non null');

```
coalesce
--------
'first non null'
```

### Tous les arguments nuls

```
PGSQL> SELECT COALESCE(NULL, NULL, NULL);

coalesce
--------
```

Lire SE FONDRE en ligne: https://riptutorial.com/fr/postgresql/topic/10576/se-fondre

# Chapitre 27: SÉLECTIONNER

## Examples

### SELECT en utilisant WHERE

Dans cette rubrique, nous nous baserons sur cette table d'utilisateurs:

```
CREATE TABLE sch_test.user_table
(
  id serial NOT NULL,
  username character varying,
  pass character varying,
  first_name character varying(30),
  last_name character varying(30),
  CONSTRAINT user_table_pkey PRIMARY KEY (id)
)

+----+------------+-----------+----------+------+
| id | first_name | last_name | username | pass |
+----+------------+-----------+----------+------+
| 1  | hello      | world     | hello    | word |
+----+------------+-----------+----------+------+
| 2  | root       | me        | root     | toor |
+----+------------+-----------+----------+------+
```

#### Syntaxe

Sélectionnez chaque chose:

```
SELECT * FROM schema_name.table_name WHERE <condition>;
```

Sélectionnez des champs:

```
SELECT field1, field2 FROM schema_name.table_name WHERE <condition>;
```

#### Exemples

```
-- SELECT every thing where id = 1
SELECT * FROM schema_name.table_name WHERE id = 1;

-- SELECT id where username = ? and pass = ?
SELECT id FROM schema_name.table_name WHERE username = 'root' AND pass = 'toor';

-- SELECT first_name where id not equal 1
SELECT first_name FROM schema_name.table_name WHERE id != 1;
```

Lire SÉLECTIONNER en ligne: https://riptutorial.com/fr/postgresql/topic/9528/selectionner

# Chapitre 28: Support JSON

## Introduction

JSON - Notation d'objet de script Java, Postgresql supporte le type de données JSON depuis la version 9.2. Il existe des fonctions et des opérateurs prédéfinis pour accéder aux données JSON. L'opérateur `->` renvoie la clé de la colonne JSON. L'opérateur `->>` renvoie la valeur de la colonne JSON.

## Examples

### Création d'une table JSON pure

Pour créer une table JSON pure, vous devez fournir un seul champ avec le type `JSONB` :

```
CREATE TABLE mytable (data JSONB NOT NULL);
```

Vous devez également créer un index de base:

```
CREATE INDEX mytable_idx ON mytable USING gin (data jsonb_path_ops);
```

À ce stade, vous pouvez insérer des données dans la table et les interroger efficacement.

### Interrogation de documents JSON complexes

Prenant un document JSON complexe dans une table:

```
CREATE TABLE mytable (data JSONB NOT NULL);
CREATE INDEX mytable_idx ON mytable USING gin (data jsonb_path_ops);
INSERT INTO mytable VALUES($$
{
    "name": "Alice",
    "emails": [
        "alice1@test.com",
        "alice2@test.com"
    ],
    "events": [
        {
            "type": "birthday",
            "date": "1970-01-01"
        },
        {
            "type": "anniversary",
            "date": "2001-05-05"
        }
    ],
    "locations": {
        "home": {
            "city": "London",
            "country": "United Kingdom"
```

```
        },
        "work": {
            "city": "Edinburgh",
            "country": "United Kingdom"
        }
    }
}
$$);
```

Requête pour un élément de niveau supérieur:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"name":"Alice"}';
```

Requête pour un élément simple dans un tableau:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"emails":["alice1@test.com"]}';
```

Requête pour un objet dans un tableau:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"events":[{"type":"anniversary"}]}';
```

Requête pour un objet imbriqué:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"locations":{"home":{"city":"London"}}}';
```

## Performance de `@>` par rapport à `->` et `->>`

Il est important de comprendre la différence de performance entre l'utilisation de `@>` , `->` et `->>` dans la partie `WHERE` de la requête. Bien que ces deux requêtes semblent être globalement équivalentes:

```
SELECT data FROM mytable WHERE data @> '{"name":"Alice"}';
SELECT data FROM mytable WHERE data->'name' = '"Alice"';
SELECT data FROM mytable WHERE data->>'name' = 'Alice';
```

la première instruction utilisera l'index créé ci-dessus alors que les deux dernières ne le seront pas, nécessitant une analyse complète de la table.

Il est toujours possible d'utiliser l'opérateur `->` lors de l'obtention des données résultantes, de sorte que les requêtes suivantes utiliseront également l'index:

```
SELECT data->'locations'->'work' FROM mytable WHERE data @> '{"name":"Alice"}';
SELECT data->'locations'->'work'->>'city' FROM mytable WHERE data @> '{"name":"Alice"}';
```

### Utilisation des opérateurs JSONb

# Création d'une base de données et d'une

# table

```
DROP DATABASE IF EXISTS books_db;
CREATE DATABASE books_db WITH ENCODING='UTF8' TEMPLATE template0;

DROP TABLE IF EXISTS books;

CREATE TABLE books (
  id SERIAL PRIMARY KEY,
  client TEXT NOT NULL,
  data JSONb NOT NULL
);
```

# Remplissage de la base de données

```
INSERT INTO books(client, data) values (
    'Joe',
    '{ "title": "Siddhartha", "author": { "first_name": "Herman", "last_name": "Hesse" } }'
),(
    'Jenny',
    '{ "title": "Dharma Bums", "author": { "first_name": "Jack", "last_name": "Kerouac" } }'
),(
    'Jenny',
    '{ "title": "100 años de soledad", "author": { "first_name": "Gabo", "last_name":
"Marquéz" } }'
);
```

Permet de tout voir à l'intérieur des livres de table:

```
SELECT * FROM books;
```

Sortie:

| id integer | client character varying | data jsonb |
|---|---|---|
| 1 | Joe | {"title": "Siddhartha", "author": {"last name": "Hesse", "first name": "Herman"}} |
| 2 | Jenny | {"title": "Dharma Bums", "author": {"last name": "Kerouac", "first name": "Jack"}} |
| 3 | Jenny | {"title": "100 años de soledad", "author": {"last name": "Marquéz", "first name": "G |

# -> opérateur renvoie des valeurs en dehors des colonnes JSON

Sélection d'une colonne:

```
SELECT client,
    data->'title' AS title
    FROM books;
```

Sortie:

| | client character varying | title jsonb |
|---|---|---|
| 1 | Joe | "Siddhartha" |
| 2 | Jenny | "Dharma Bums" |
| 3 | Jenny | "100 años de soledad" |

Sélection de 2 colonnes:

```
SELECT client,
   data->'title' AS title, data->'author' AS author
   FROM books;
```

Sortie:

| client character varying | title jsonb | author jsonb |
|---|---|---|
| Joe | "Siddhartha" | {"last_name": "Hesse", "first_name": "Herman"} |
| Jenny | "Dharma Bums" | {"last name": "Kerouac", "first name": "Jack"} |
| Jenny | "100 años de soledad" | {"last name": "Marquéz", "first name": "Gabo"} |

## -> vs ->>

L'opérateur `->` renvoie le type JSON d'origine (qui peut être un objet), tandis que `->>` renvoie le texte.

# Retourne des objets NESTED

Vous pouvez utiliser le `->` pour renvoyer un objet imbriqué et ainsi enchaîner les opérateurs:

```
SELECT client,
   data->'author'->'last_name' AS author
   FROM books;
```

Sortie:

| client character varying | author jsonb |
|---|---|
| Joe | "Hesse" |
| Jenny | "Kerouac" |
| Jenny | "Marquéz" |

# Filtration

Sélectionnez des lignes en fonction d'une valeur dans votre JSON:

```
 SELECT
 client,
 data->'title' AS title
 FROM books
  WHERE data->'title' = '"Dharma Bums"';
```

Notez que WHERE utilise `->` donc nous devons comparer à JSON `'"Dharma Bums"'`

Ou nous pourrions utiliser `->>` et comparer à `'Dharma Bums'`

Sortie:

| client<br>character varying | title<br>jsonb |
|---|---|
| Jenny | "Dharma Bums" |

# Filtrage imbriqué

Recherchez des lignes en fonction de la valeur d'un objet JSON imbriqué:

```
SELECT
 client,
 data->'title' AS title
 FROM books
  WHERE data->'author'->>'last_name' = 'Kerouac';
```

Sortie:

| client<br>character varying | title<br>jsonb |
|---|---|
| Jenny | "Dharma Bums" |

# Un exemple du monde réel

```
CREATE TABLE events (
  name varchar(200),
  visitor_id varchar(200),
  properties json,
  browser json
);
```

Nous allons stocker les événements dans cette table, comme les pages vues. Chaque événement a des propriétés, qui peuvent être n'importe quoi (par exemple, la page en cours) et envoie également des informations sur le navigateur (comme le système d'exploitation, la résolution d'écran, etc.). Les deux sont totalement gratuits et pourraient changer avec le temps (car nous pensons à des choses supplémentaires à suivre).

```
INSERT INTO events (name, visitor_id, properties, browser) VALUES
(
  'pageview', '1',
```

```
  '{ "page": "/" }',
  '{ "name": "Chrome", "os": "Mac", "resolution": { "x": 1440, "y": 900 } }'
),(
  'pageview', '2',
  '{ "page": "/" }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1920, "y": 1200 } }'
),(
  'pageview', '1',
  '{ "page": "/account" }',
  '{ "name": "Chrome", "os": "Mac", "resolution": { "x": 1440, "y": 900 } }'
),(
  'purchase', '5',
  '{ "amount": 10 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1024, "y": 768 } }'
),(
  'purchase', '15',
  '{ "amount": 200 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 } }'
),(
  'purchase', '15',
  '{ "amount": 500 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 } }'
);
```

Maintenant, permet de sélectionner tout:

```
SELECT * FROM events;
```

Sortie:

| name<br>character varying(200) | visitor_id<br>character varying(200) | properties<br>json | browser<br>json |
|---|---|---|---|
| pageview | 1 | { "page": "/" } | { "name": "Chrome", "os": "Mac", "resolution |
| pageview | 2 | { "page": "/" } | { "name": "Firefox", "os": "Windows", "resol |
| pageview | 1 | { "page": "/account" } | { "name": "Chrome", "os": "Mac", "resolution |
| purchase | 5 | { "amount": 10 } | { "name": "Firefox", "os": "Windows", "resol |
| purchase | 15 | { "amount": 200 } | { "name": "Firefox", "os": "Windows", "resol |
| purchase | 15 | { "amount": 500 } | { "name": "Firefox", "os": "Windows", "resol |

# Opérateurs JSON + fonctions d'agrégation PostgreSQL

En utilisant les opérateurs JSON, combinés aux fonctions d'agrégation PostgreSQL traditionnelles, nous pouvons tirer ce que nous voulons. Vous avez la pleine puissance d'un SGBDR à votre disposition.

- Permet de voir l'utilisation du navigateur:

```
SELECT browser->>'name' AS browser,
  count(browser)
  FROM events
  GROUP BY browser->>'name';
```

Sortie:

| browser text | count bigint |
|---|---|
| Firefox | 4 |
| Chrome | 2 |

- Total des revenus par visiteur:

```
SELECT visitor_id, SUM(CAST(properties->>'amount' AS integer)) AS total
FROM events
WHERE CAST(properties->>'amount' AS integer) > 0
GROUP BY visitor_id;
```

Sortie:

| visitor_id character varying(200) | total bigint |
|---|---|
| 5 | 10 |
| 15 | 700 |

- Résolution d'écran moyenne

```
SELECT AVG(CAST(browser->'resolution'->>'x' AS integer)) AS width,
  AVG(CAST(browser->'resolution'->>'y' AS integer)) AS height
FROM events;
```

Sortie:

| width numeric | height numeric |
|---|---|
| 1397.3333333333333333 | 894.666666666666667 |

Plus d'exemples et de documentation ici et ici .

Lire Support JSON en ligne: https://riptutorial.com/fr/postgresql/topic/1034/support-json

# Chapitre 29: Types de données

## Introduction

PostgreSQL dispose d'un riche ensemble de types de données natifs à la disposition des utilisateurs. Les utilisateurs peuvent ajouter de nouveaux types à PostgreSQL en utilisant la commande CREATE TYPE.

https://www.postgresql.org/docs/9.6/static/datatype.html

## Examples

### Types numériques

| prénom | Taille de stockage | La description | Gamme |
|---|---|---|---|
| `smallint` | 2 octets | entier à petite portée | -32768 à +32767 |
| `integer` | 4 octets | choix ypical pour entier | -2147483648 à +2147483647 |
| `bigint` | 8 octets | entier à grande portée | -9223372036854775808 à +9223372036854775807 |
| `decimal` | variable | précision spécifiée par l'utilisateur, exacte | jusqu'à 131072 chiffres avant le point décimal; jusqu'à 16383 chiffres après le point décimal |
| `numeric` | variable | précision spécifiée par l'utilisateur, exacte | jusqu'à 131072 chiffres avant le point décimal; jusqu'à 16383 chiffres après le point décimal |
| `real` | 4 octets | précision variable, inexacte | Précision de 6 chiffres décimaux |
| `double precision` | 8 octets | précision variable, inexacte | Précision de 15 chiffres décimaux |
| `smallserial` | 2 octets | petit entier auto-incrémentant | 1 à 32767 |
| `serial` | 4 octets | entier auto-incrémentant | 1 à 2147483647 |

| prénom | Taille de stockage | La description | Gamme |
|---|---|---|---|
| `bigserial` | 8 octets | grand entier auto-incrémentant | 1 à 9223372036854775807 |
| `int4range` | | Gamme d'entier | |
| `int8range` | | Gamme de bigint | |
| `numrange` | | Gamme de numérique | |

### Types date / heure

| prénom | Taille de stockage | La description | Valeur faible | Haute valeur | Résolution |
|---|---|---|---|---|---|
| `timestamp` (sans fuseau horaire) | 8 octets | la date et l'heure (pas de fuseau horaire) | 4713 avant JC | 294276 AD | 1 microseconde / 14 chiffres |
| `timestamp` (avec fuseau horaire) | 8 octets | à la fois la date et l'heure, avec le fuseau horaire | 4713 avant JC | 294276 AD | 1 microseconde / 14 chiffres |
| `date` | 4 octets | date (pas d'heure de la journée) | 4713 avant JC | 5874897 AD | Un jour |
| `time` (sans fuseau horaire) | 8 octets | heure du jour (pas de date) | 00:00:00 | 24:00:00 | 1 microseconde / 14 chiffres |
| `time` (avec fuseau horaire) | 12 octets | temps de la journée seulement, avec fuseau horaire | 00: 00: 00 + 1459 | 24h00: 00-1459 | 1 microseconde / 14 chiffres |
| `interval` | 16 octets | intervalle de temps | -178000000 ans | 178000000 ans | 1 microseconde / 14 chiffres |
| `tsrange` | | plage d'horodatage sans fuseau horaire | | | |
| `tstzrange` | | plage | | | |

| prénom | Taille de stockage | La description | Valeur faible | Haute valeur | Résolution |
|---|---|---|---|---|---|
| | | d'horodatage avec fuseau horaire | | | |
| `daterange` | | gamme de date | | | |

### Types géométriques

| prénom | Taille de stockage | La description | Représentation |
|---|---|---|---|
| `point` | 16 octets | Point sur un avion | (x, y) |
| `line` | 32 octets | Ligne infinie | {ABC} |
| `lseg` | 32 octets | Segment de ligne fini | ((x1, y1), (x2, y2)) |
| `box` | 32 octets | Boîte rectangulaire | ((x1, y1), (x2, y2)) |
| `path` | 16 + 16n octets | Chemin fermé (similaire au polygone) | ((x1, y1), ...) |
| `path` | 16 + 16n octets | Chemin ouvert | [(x1, y1), ...] |
| `polygon` | 40 + 16n octets | Polygone (similaire au chemin fermé) | ((x1, y1), ...) |
| `circle` | 24 octets | Cercle | <(x, y), r> (centre et rayon) |

### Types d'adresse réseau

| prénom | Taille de stockage | La description |
|---|---|---|
| `cidr` | 7 ou 19 octets | Réseaux IPv4 et IPv6 |
| `inet` | 7 ou 19 octets | Hôtes et réseaux IPv4 et IPv6 |
| `macaddr` | 6 octets | Adresses MAC |

### Types de caractères

| prénom | La description |
| --- | --- |
| `character varying(n)` , `varchar(n)` | longueur variable avec limite |
| `character(n)` , caractère `char(n)` | longueur fixe, matelassé |
| `text` | longueur illimitée variable |

### Tableaux

Dans PostgreSQL, vous pouvez créer des tableaux de tout type intégré, défini par l'utilisateur ou enum. Par défaut, il n'y a pas de limite à un tableau, mais vous *pouvez le* spécifier.

## Déclarer un tableau

```
SELECT integer[];
SELECT integer[3];
SELECT integer[][];
SELECT integer[3][3];
SELECT integer ARRAY;
SELECT integer ARRAY[3];
```

## Créer un tableau

```
SELECT '{0,1,2}';
SELECT '{{0,1},{1,2}}';
SELECT ARRAY[0,1,2];
SELECT ARRAY[ARRAY[0,1],ARRAY[1,2]];
```

## Accéder à un tableau

Par défaut, PostgreSQL utilise une convention de numérotation à base unique pour les tableaux, c'est-à-dire qu'un tableau de n éléments commence par `array[1]` et se termine par `array[n]` .

```
--accesing a spefific element
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1] FROM arr;

int_arr
---------
        0
(1 row)

--sclicing an array
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1:2] FROM arr;

int_arr
---------
    {0,1}
(1 row)
```

## Obtenir des informations sur un tableau

```
--array dimensions (as text)
with arr as (select ARRAY[0,1,2] int_arr) select array_dims(int_arr) from arr;

array_dims
------------
       [1:3]
(1 row)

--length of an array dimension
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT array_length(int_arr,1) FROM arr;

 array_length
 --------------
              3
 (1 row)

--total number of elements across all dimensions
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT cardinality(int_arr) FROM arr;

 cardinality
 -------------
             3
 (1 row)
```

## Fonctions de tableau

sera ajouté

Lire Types de données en ligne: https://riptutorial.com/fr/postgresql/topic/8976/types-de-donnees

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Démarrer avec postgresql | a_horse_with_no_name, Alison S, AndrewCichocki, Ben, Ben H , bignose, Community, Dakota Wagner, DeadEye, Demircan Celebi, Dmitri Goldring, e4c5, , jasonszhao, Kirk Roybal, Marek Skiba, Mokadillion, Patrick, user_0 |
| 2 | Accéder aux données par programmation | AstraSerg, brichins, greg, Laurenz Albe |
| 3 | Astuce Postgres | Ben H, skj123, user_0, YCF_L |
| 4 | Commentaires dans postgresql | Ben, KIRAN KUMAR MATAM |
| 5 | Création de table | e4c5, Jefferson, KIM, leeor, Patrick |
| 6 | Dates, horodatages et intervalles | KIM, Nuri Tasdemir, Patrick, Tom Gerken |
| 7 | Déclencheurs d'événement | Ben H, Tajinder, Udlei Nati |
| 8 | Exporter l'en-tête et les données de la table de base de données PostgreSQL dans un fichier CSV | Vao Tsun, wOwhOw |
| 9 | Expressions de table communes (WITH) | Daniel Lyons, Jakub Fedyczak, Kevin Sylvestre |
| 10 | EXTENSION dblink et postgres_fdw | Riya Bansal, YCF_L |
| 11 | Fonctions cryptographiques postgres | Ben H, skj123 |
| 12 | Fonctions d'agrégat | Alison S, joseph, Kirill Sokolov, Patrick |
| 13 | Fonctions de | chalitha geekiyanage, mnoronha, Udlei Nati |

| | | |
|---|---|---|
| | déclenchement et de déclenchement | |
| 14 | Fonctions de fenêtre | mnoronha, Vao Tsun |
| 15 | Gestion des rôles | Ben, Ben H, bilelovitch, Blackus, Daniel Lyons, e4c5, greg, KIM, Laurenz Albe, mnoronha, Reboot |
| 16 | Haute disponibilité PostgreSQL | gpdude_, Patrick |
| 17 | Héritage | evuez |
| 18 | INSÉRER | chalitha geekiyanage, e4c5, gpdude_, KIM, lamorach, leeor, Nathaniel Waisbrot, Patrick, Vao Tsun |
| 19 | METTRE À JOUR | frlan, leeor |
| 20 | Programmation avec PL / pgSQL | AndrewCichocki, Ben H, Goerman, Laurenz Albe, Vao Tsun |
| 21 | Rechercher longueur de chaîne / longueur de caractère | Mohamed Navas |
| 22 | Requêtes récursives | Ben H |
| 23 | Sauvegarde et restauration | ankidaemon, Ben H, Daniel Lyons, e4c5, Laurel, mnoronha |
| 24 | Script de sauvegarde pour une base de données de production | bilelovitch |
| 25 | Se connecter à PostgreSQL à partir de Java | Laurenz Albe |
| 26 | SE FONDRE | Mokadillion |
| 27 | SÉLECTIONNER | YCF_L |
| 28 | Support JSON | Clodoaldo Neto, commonSenseCode, jgm, KIRAN KUMAR MATAM, mnoronha, Peter Krauss |
| 29 | Types de données | Ben H, user_0 |