BASE DE DONNEES POSTGRESQL

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Base de données PostgreSql

Installation avec Linux

Si vous utilisez Ubuntu de version supérieure à 12.04 alors vous avez peut-être déjà installé PostgreSQL lors de l'installation d'Ubuntu (faire **psql -V** pour voir si vous l'avez déjà) .

Dans le cas contraire, on utilise le gestionnaire de paquet de Ubuntu qui va faire tout le travail pour nous. Pour cela on ouvre une console et on tape la commande suivante :

sudo apt-get install *postgresql*

Il va vous demander de saisir votre mot de passe utilisateur pour continuer l'installation. Puis télécharger les paquets, et procéder à l'installation. Une fois l'installation achevée, on va vérifier qu'on possède bien la dernière version en tapant la commande suivante dans la console :

psql -V

Base de données PostgreSql

Installation avec Windows

Pour l'installation sous Windows, il faut le télécharger sur la page *http://www.postgresql.org/download/windows*.

Une fois le fichier exécutable téléchargé, il suffit de le lancer et de suivre les étapes pas à pas:

- Choisir le répertoire d'installation de PostgreSQL : On laisse l'emplacement par défaut (C:\Program Files\PostgreSQL\version)
- Choisir le répertoire où les données seront stockées : on laisse aussi l'emplacement par défaut
- Définir le mot de passe administrateur de la base : pour faire simple, on va mettre "*imie*"
- Spécifier le port d'écoute : on laisse le port par défaut (port 5432)
- Localisation : on laisse la localisation par défaut du système
- On valide enfin l'installation
- Une fois l'installation achevée, l'installer va vous proposer de lancer Stack Builder qui permet d'installer des outils supplémentaires. Pour notre cours, nous n'en avons pas besoin.

Base de données PostgreSql

Arrêter ou Redémarrer Postgres avec Lunix

Après l'installation Postgresql est déjà démarré. Un des cas de l'arret et redémarrage est sur le fait qu'on a fait quelques modifications sur les paramètres.

/etc/init.d# service postgresql {start | stop }

Base de données PostgreSql

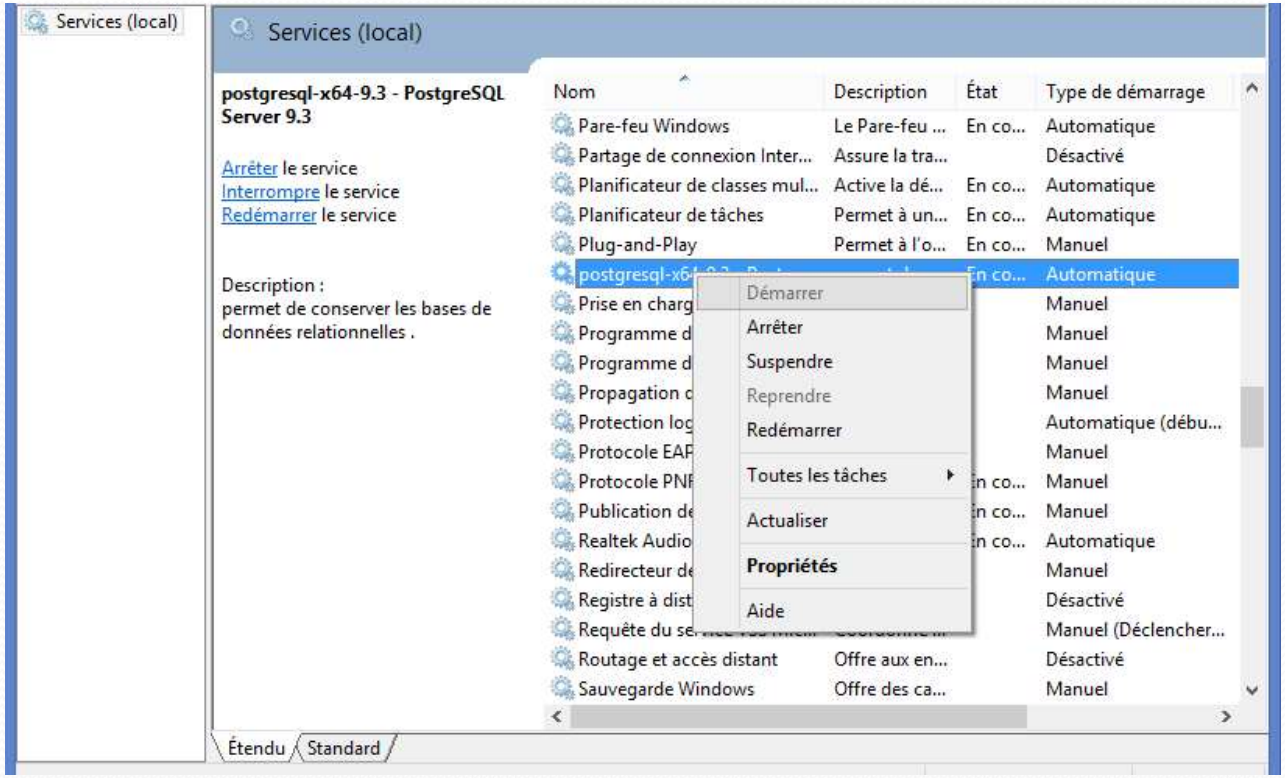
Arrêter ou Redémarrer Postgres avec Windows

Utilisez le raccourci clavier touche *Windows + R* puis tapez *services.msc*

Cherchez le service "*PostgreSQL Server*" et cliquez à droite:

-Arreter pour arrêter le service

-Redemarrer pour redémarrer le service



Base de données PostgreSql

Création d'une base de données

Il est possible de créer une base de données à partir de la commande **createdb**

> Createdb –U nomUtilisateur nomBaseDeDonnées

Exemple : création de la base de donnée ma_base via l'utilisateur postgres

> Createdb –U postgres ma_base

Base de données PostgreSql

Accéder à une base de données

Pour accéder à une base de données à partir de l'*invite commande*' on utilise la commande **psql**

Prenons comme exemple la base de donnée ma_base:

> psql -U postgres ma_base

Base de données PostgreSql

Commandes internes (1/2)

Commande	Description
\c (ou \connect) [nom_base [nom_utilisateur] [hôte] [port]	Établie une nouvelle connexion à un serveur PostgreSQL Exemple : **\c maboutique;**
\copy (*requête*) to *nomfichier*	Copie le résultat de la requête vers le fichier ***nomfichier*** Exemple : *\copy (select id, substr(nom,1,20) as nom from client) to c:/essaipsql.txt*
Attention ! ne mettez pas ";" à la fin de la ligne commande sinon le nom du fichier se terminera par ";").	
\i *nomfichier*	Lit l'entrée à partir du fichier ***nomfichier*** et l'exécute comme si elle avait été saisie sur le clavier.

Base de données PostgreSql

Commandes internes (2/2)

Commande	Description
\l (ou \list)	Liste les noms de toutes les bases de données du serveur.
\o ***nomfichier***	Sauvegarde les résultats des requêtes dans le fichier **nomfichier** .
\o	L'affichage de la requête est redirigé vers la sortie standard.
\d nomTable	Affiche la structure de la table **nomTable** .
\d	Affiche la liste de toutes les tables dans la base de données en cours.

Base de données PostgreSql

Espace de tables

Un espace de tables est un répertoire d'un système de fichiers, dans lequel PostgreSql écrit les fichiers des tables et des index. Par défaut , il dispose d'un espace de tables, situé dans le répertoire du groupe de base de données.
Etapes à suivre pour la création d'un espace de table:

- Créer un répertoire par exemple (H:\TEST\EspaceTables)
- -Utiliser la commande CREATE TABLESPACE :

CREATE TABLESPACE nomTablespace LOCATION 'repertoire';

Exemple :
Create tablespace MonEspaceTables location 'H:/TEST/EspaceTables',

- Créer une base de données dans la tablespace :

CREATE DATABASE nomBasedeDonnées TABLESPACE nomTablespace;

Renommer un espace de tables:
ALTER TABLESPACE anciennomtablespace RENAME to noveaunomtablespace

Modifier l'espace table de'une base de données
ALTER DATABASE nomBasedeDonnées set TABLESPACE nomTableSpace
Supprimer un espace de tables:

DROP TABLESPACE [IF EXISTS] nomtablespace
Remarque: le repertoire doit être vide avant la suppresion n de l'espace de tables.

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Langage SQL

Créer une table

Vous pouvez créer une table en spécifiant le nom de la table, suivi du nom de toutes les colonnes et de leur type .

Par exemple , créons la table nommée ***client***:

CREATE TABLE client(id serial primary key ,
nomClient varchar(80), adresseClient varchar(150)
) tablespace MesTables;

Les options:

- ✓ *serial* permet d'incrémenter la valeur de l'attribut ***id*** à chaque insertion. Elle correspond à auto_increment pour certaine base de données. Pour changer la valeur départ : *Alter sequence NomSequence restart ValeurDepart*
- ✓ *primary key* précise que ***id*** est la clé primaire de la table ***client***
- ✓ *TableSpace* permet de stocker la table créée dans un répertoire défini par *MesTables*.

Créer une séquence

Une séquence correspond à un compteur qui peut être manipulé avec quelques fonctions; notamment la fonction *nextval()* qui permet d'incrémenter et de récupérer la valeur du compteur .

Pour la création:

CREATE SEQUENCE nomSequence
[INCREMENT [BY] increment]
[START [WITH] debut]
[MINVALUE valeurmin]
[MAXVALUE valeurmax]
[CYCLE]

CYCLE autorise la séquence à revenir à la valeur minimale une fois que la valeur maximale est atteinte.

Exemple:

Create sequence numero_seq;
Create table produit (id integer default nextval('numero_seq'::regclass),
designation varchar(100));

Où regclass est un type d'identificateur d'objet

Cet exemple équivaut à :

Create table produit (id serial , designation varchar(150));

Langage SQL

Modifier une séquence

La modification d'une séquence permet par exemple une réinitialisation de la séquence.

ALTER SEQUENCE nomSequence
[INCREMENT [BY] increment]
[MINVALUE valeurmin]
[MAXVALUE valeurmax]
[RESTART [WITH] debut]

Supprimer une séquence

DROP SEQUENCE nomSequence

Langage SQL

Clés étrangères (1/3)

Soient les tables **client** et **commande** . Il s'agit maintenant de s'assurer que personne n'insère de ligne dans la table commande qui ne corresponde à une entrée dans la table Client.

CREATE TABLE commande(id serial primary key, client_id integer references client(id), dateCommande date));

A cet effet , ***toute insertion*** d'enregistrement dans la table commande provoquera une erreur si la valeur de l'attribut client_id n'existe pas dans la table client.

Langage SQL

Remplir un table

L'instruction **INSERT** est utilisé pour remplir une table.

Prenons comme exemple la table client:

INSERT INTO Client (nomClient, adresseClient) VALUES ('Joé','Saint-Nazaire'), ('Richard','Angres');

Il est aussi possible de aussi remplir une Table par une autre table de même structure:

INSERT INTO Client (nomClient, adresseClient) (Select nom,Adreese from Fournisseur);

Langage SQL

Ajouter une contrainte

Pour ajouter une contrainte, la syntaxe de contrainte de table est utilisée.

Exemples :

ALTER TABLE produits ADD CHECK (nom <> '');

ALTER TABLE produits ADD CONSTRAINT autre_nom UNIQUE (no_produit);

ALTER TABLE produits ADD FOREIGN KEY (id_groupe_produits) REFERENCES groupe_produits;

Pour ajouter une contrainte ***NOT NULL***, qui ne peut pas être écrite sous forme d'une contrainte de table, la syntaxe suivante est utilisée :

ALTER TABLE produits ALTER COLUMN no_produit SET NOT NULL;

La contrainte étant immédiatement vérifiée, les données de la table doivent satisfaire la contrainte avant qu'elle ne soit ajoutée.

Langage SQL

Supprimer une contrainte

La suppression se fait par son nom si elle a été explicitement nommée. Dans le cas contraire, le système engendre et attribue un nom qu'il faut découvrir à partir de commande **\d *table*** de **psql.**

La commande est :

ALTER TABLE produits DROP CONSTRAINT un_nom;

La contrainte ***NOT NULL*** n'a pas de nom, alors la suppression se fait directement sans référence à un nom.

ALTER TABLE produits ALTER COLUMN no_produit DROP NOT NULL;

Langage SQL

Modifier la valeur par défaut d'une colonne

La commande de définition d'une nouvelle valeur par défaut de colonne ressemble à celle-ci :

ALTER TABLE produits ALTER COLUMN prix SET DEFAULT 7.77;

Pour retirer toute valeur par défaut, on écrit :

ALTER TABLE produits ALTER COLUMN prix DROP DEFAULT;

Langage SQL

Modifier le type de données d'une colonne:

ALTER TABLE produits ALTER COLUMN prix TYPE numeric(10,2);

Renommer une colonne:

ALTER TABLE produits RENAME COLUMN no_produit TO numero_produit;

Renommer une table:

ALTER TABLE produits RENAME TO Articles;

Fenetrage (1/3)

Une *fonction de fenêtrage* effectue un calcul sur un jeu d'enregistrements liés d'une certaine façon à l'enregistrement courant.

On peut les rapprocher des calculs réalisables par une fonction d'agrégat sans regrouper les enregistrements traités.

Exemple 1:

SELECT nomdep, noemp, salaire, avg(salaire) OVER (PARTITION BY nomdep) FROM salaireemp;

nomdep	noemp	salaire	avg
develop	11	5200	5020.00
develop	7	4200	5020.00
develop	9	4500	5020.00
develop	8	6000	5020.00
develop	10	5200	5020.00
personnel	5	3500	3700.00
personnel	2	3900	3700.00
ventes	3	4800	4866.66
ventes	1	5000	4866.66
ventes	4	4800	4866.66

(10 rows)

Fenetrage (2/3)

Exemple 2:

SELECT nomdep, noemp, salaire, rank() OVER (PARTITION BY nomdep ORDER BY salaire DESC)
FROM salaireemp;

nomdep	noemp	salaire	rank
develop	8	6000	1
develop	10	5200	2
develop	11	5200	2
develop	9	4500	4
develop	7	4200	5
personnel	2	3900	1
personnel	5	3500	2
ventes	1	5000	1
ventes	4	4800	2
ventes	3	4800	2

(10 rows)

On remarque que la fonction ***rank*** () produit un rang numérique dans la partition de l'enregistrement pour chaque valeur différente de l'ORDER BY, dans l'ordre défini par la clause ORDER BY.

Langage SQL

Fenetrage (3/3)

Exemple 3:

SELECT salaire, sum(salaire) OVER () FROM salaireemp;

salaire	sum
5200	47100
5000	47100
3500	47100
4800	47100
3900	47100
4200	47100
4500	47100
4800	47100
6000	47100
5200	47100

(10 rows)

Exemple 4:

SELECT salaire, sum(salaire) OVER (ORDER BY salaire) FROM salaireemp;
salaire| sum

salaire	sum
3500	3500
3900	7400
4200	11600
4500	16100
4800	25700
4800	25700
5000	30700
5200	41100
5200	41100
6000	47100

(10 rows)

Langage SQL

Tables jointes (1/2)

Les jointures internes (**inner**), externes (**outer**) et croisées (**cross**) sont disponibles en Postgres

Soient t1 et t2 deux tables :

Table t1

no	nom
1	a
2	b
3	c

Table t2

no	valeur
1	xxx
3	yyy
5	zzz

*SELECT * FROM t1 CROSS JOIN t2;*

*SELECT * FROM t1 ,t2;*

no	nom	no	valeur
1	a	1	xxx
1	a	3	yyy
1	a	5	zzz
2	b	1	xxx
2	b	3	yyy
2	b	5	zzz
3	c	1	xxx
3	c	3	yyy
3	c	5	zzz

(9 rows)

*SELECT * FROM t1 INNER JOIN t2 on t1.no=t2.no;*

*SELECT * FROM t1 ,t2 WHERE t1.no=t2.no;*

no	nom	valeur
1	a	xxx
3	c	yyy

Tables jointes (2/2)

*SELECT * FROM t1 LEFT JOIN t2 ON t1.no = t2.no ;*

no	nom	no	valeur
1	a	1	xxx
2	b		
3	c	3	yyy

(3 rows)

*SELECT * FROM t1 RIGHT JOIN t2 ON t1.no = t2.no;*

no	nom	no	valeur
1	a	1	xxx
3	c	3	yyy
		5	zzz

(3 rows)

*SELECT * FROM t1 FULL JOIN t2 ON t1.no = t2.no;*

no	nom	no	valeur
1	a	1	xxx
2	b		
3	c	3	yyy
		5	zzz

(4 rows)

Explain

Langage SQL

La commande EXPLAIN permet d'étudier le comportement d'une requête et notamment les différentes méthodes utilisées par Postgresql pour acceder aux données:

EXPLAIN [ANALYSE] requete

L'option ANALYSE execute reellement la requete et donne le temps d'execution.

Exemple:

explain analyse select * from article where id=91;

QUERY PLAN

Index Scan using article_pkey on article (cost=0.28..8.30 rows=1 width=57) (actual time=0.024..0.025 rows=1 loops=1)

Index Cond: (id = 91)

Total runtime: 0.065 ms

(3 lignes)

VACUUM

La commande VACUUM est un outil de nettoyage de la base de données. Lors des opération normales, par exemple les mises à jour de ligne avec UPDATE, ou les suppressions avec DELETE, les lignes ne sont pas libérées. Le nettoyage permet de reutiliser ces lignes.

VACUUM [FULL]

-FULL permet en plus de compacter les tables.

REINDEX

La commande REINDEX permet de maintenir un ou plusieurs index, losqu'ils sont corrompus ou simplement s'il existe des données inutiles dans l'index:

REINDEX {INDEX | TABLE | DATABASE } NOM

Exemples:

Reindex index idx_article_code

Reindex table commande;

Reindex database maboutique;

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Système de règles

Regles sur INSERT, UPDATEet DELETE

Ces règles se font par la commande ***CREATE RULE***

Syntaxe:

CREATE RULE nom_rule AS ON evenement TO nomTable [WHERE condition] DO [ALSO|INSTEAD] [action | NOTHING];

Paramètres:

nom_rule : Le nom d'une règle à créer.

evenement: C'est l'un des SELECT, INSERT, UPDATE ou DELETE.

nomTable: Le nom de la Table ou de la Vue où s'applique la règle.

condition: Toute expression conditionnelle SQL (retour booléenne).

ALSO : Indique que les commandes doivent être exécutées **en plus de** la commande d'origine.

INSTEAD : Indique au contraire que les commandes doivent être exécutées **à la place de** la commande d'origine.

Si ni *ALSO* ni *INSTEAD* ne sont spécifiés, *ALSO* est la valeur par défaut.

Action: La ou les commandes qui composent l'action de règle. Les commandes SELECT, INSERT, UPDATE et DELETE sont acceptées.

NOTHING: aucune action à faire.

Système de règles

Exemple 1:

On veut enregistrer dans une table Article_log les traces de modification de prix effectuées dans la table Article.

```
----------création de la table article_log----------
CREATE TABLE article_log (    id integer,designation
varchar(250),nouveau_prix float,ancien_prix float,
nom_acteur text,heure  timestamp);

----------création de la règle article_rule----------
CREATE RULE article_rule  AS ON UPDATE TO article
WHERE NEW.prix <> OLD.prix     DO INSERT INTO
article_log VALUES
(NEW.id,NEW.designation,NEW.prix,OLD.prix
,current_user,current_timestamp);
```

Système de règles

Exemple 2:

Dans cet exemple on joue le rôle d' une contrainte d' intégrité sur une clé étrangère:

```
create or replace rule verifcommande as on insert to
lignecommande
where (not exists (select commande.id from commande where
commande.id=NEW.commande_id))  DO INSTEAD  nothing;
```

ou

```
create or replace rule verifcommande as on insert to
lignecommande
where (select commande.id from commande where
commande.id=NEW.commande_id) is null   DO INSTEAD  (select
'Attention!  Client inexistent' as Message);
```

Système de règles

Afficher les règles

Pour afficher les regles, il suffit de voir la structure de la table ui les contient.

Exemple:

```
\d  ligneCommande
=>   Règles :
            verifcommande AS
            ON INSERT TO lignecommande
           WHERE NOT (EXISTS ( SELECT commande.id
                FROM commande
               WHERE commande.id = new.commande_id)) DO INSTEAD NOTHING
```

Supprimer une règle

Syntaxe:

DROP RULE [IF EXISTS] nom ON nom_table [CASCADE | RESTRICT]

Exemple:

```
Drop  rule verifcommande on LigneCommande;
```

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Héritage et Partitionnement

Héritage (1/2)

L'héritage est un concept issu des bases de données orientées objet. Il est réalisé par la commande INHERITS.
Soient deux tables : une table villes et une table capitales. Les capitales étant également des villes, il est intéressant d'avoir la possibilité d'afficher implicitement les capitales lorsque les villes sont listées.

```
CREATE TABLE villes (
nom text,
population real,
altitude int -- (en pied)
);
CREATE TABLE capitales (
etat char(2)
) INHERITS (villes);
```

Dans ce cas, une ligne de **capitales** *hérite* de toutes les colonnes (*nom*, *population* et *altitude*) de son *parent* **villes**.

Héritage et Partitionnement

Héritage (2/2)

Exemple 2:

SELECT nom, altitude FROM villes WHERE altitude > 500;

nom	altitude
Las Vegas	2174
Mariposa	1953
Madison	845

(3 rows)

➢La requête ci-dessus fournit un exemple d'extraction des noms de toutes les villes, en incluant les capitales des états, situées à une altitude de plus de 500 pieds

Exemple 3:

SELECT nom, altitude FROM ONLY villes WHERE altitude > 500;

nom	altitude
Las Vegas	2174
Mariposa	1953

(2 rows)

- Ici, ONLY indique que la requête ne doit être exécutée que sur la table villes seulement sans ses héritiers.

Héritage et Partitionnement

Partitionnement (1/4)

Le partitionnement fait référence à la division d'une table logique volumineuse en plusieurs parties physiques plus petites.
Toutefois, le partitionnement doit être envisagé si la taille de la table peut être amenée à dépasser la taille de la mémoire physique du serveur Postgres réalise le partitionnement à travers l'héritage de tables.

Chaque partition doit être créée comme une table enfant d'une unique table parent.

La table parent peut être vide, dans ce cas, elle n'existe que pour représenter l'ensemble complet des données.

Héritage et Partitionnement

Partitionnement (2/4)

Pour partionner une table, la procédure est la suivante :

- Créer la table « maître ». C'est de celle-ci qu'héritent toutes les partitions.
- Les contraintes de vérification ne doivent être définies sur cette table que si elles sont appliquées à toutes les partitions.
- Créer plusieurs tables enfants qui héritent chacune de la table maître. Ces tables enfants sont appelées partitions.
- Ajouter les contraintes de tables aux tables de partitions pour définir les valeurs des clés autorisées dans chacune.

Héritage et Partitionnement

Partitionnement (3/4)

Soit la base de données d'une grande fabrique de glaces. La compagnie mesure le pic de température journalier ainsi que les ventes de glaces dans chaque région.

1. ***Créons la table maitre*** :

```
CREATE TABLE mesure (
 id_ville       int not null,
 date_trace     date not null,
 temperature    int,
ventes          int);
```

2. ***Créons les tables partitions avec leurs containtes***:

- CREATE TABLE mesure_a2004m02 (CHECK (date_trace >= DATE '2004-02-01' AND date_trace < DATE '2004-03-01')) INHERITS (mesure);
- CREATE TABLE mesure_a2004m03 (CHECK (date_trace >= DATE '2004-03-01' AND date_trace < DATE '2004-04-01')) INHERITS (mesure);
- ...
- CREATE TABLE mesure_a2005m11 (CHECK (date_trace >= DATE '2005-11-01' AND date_trace < DATE '2005-12-01')) INHERITS (mesure);
- CREATE TABLE mesure_a2005m12 (CHECK (date_trace >= DATE '2005-12-01' AND date_trace < DATE '2006-01-01')) INHERITS (mesure);
- CREATE TABLE mesure_a2006m01 (CHECK (date_trace >= DATE '2006-01-01' AND date_trace < DATE '2006-02-01')) INHERITS (mesure);

Héritage et Partitionnement

Partitionnement (4/4)

3.Des index sur les colonnes clés sont probablement nécessaires :

- CREATE INDEX mesure_a2004m02_date_trace ON mesure_a2004m02 (date_trace);
- CREATE INDEX mesure_a2004m03_date_trace ON mesure_a2004m03 (date_trace);
- ...
- CREATE INDEX mesure_a2005m11_date_trace ON mesure_a2005m11 (date_trace);
- CREATE INDEX mesure_a2005m12_date_trace ON mesure_a2005m12 (date_trace);
- CREATE INDEX mesure_a2006m01_date_trace ON mesure_a2006m01 (date_trace);

On peut supprimer les anciennes partitions à l'aide de DROP TABLE:

```
DROP TABLE mesure_a2003m02;
```

Une autre option, souvent préférable, consiste à supprimer la partition de la table partitionnée mais de conserver l'accès à la table en tant que telle :

```
ALTER TABLE mesure_a2003m02 NO INHERIT mesure;
```

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Schémas

Une base de données contient un ou plusieurs ***schémas***, qui, eux, contiennent des tables.

Le même nom de table peut être utilisé dans différents schémas sans conflit ; par exemple, schema1 et schema2 peuvent tous les deux contenir une table nommée ma_table.

Il existe plusieurs raisons d'utiliser les schémas :

- autoriser de nombreux utilisateurs à utiliser une base de données sans interférences entre eux ;
- organiser les objets de la base de données en groupes logiques afin de faciliter leur gestion .
- éviter les collisions avec les noms d'autres objets

Schémas

Créer un schéma

Pour créer un schéma, on utilise la commande ***CREATE SCHEMA***.

Exemple : *CREATE SCHEMA mon_schema;*

Pour créer ou accéder aux tables d'un schéma, on écrit :
schema.table

Pour créer une table dans le ***nouveau*** schéma, on utilise

CREATE TABLE mon_schema.ma_table (...);

Pour effacer un schéma vide (tous les objets qu'il contient ont été supprimés), on utilise

DROP SCHEMA mon_schema;

Pour effacer un schéma et les objets qu'il contient, on utilise

DROP SCHEMA mon_schema CASCADE;

Schémas

Chemin de parcours des schémas

- Le chemin de recherche courant est affiché à l'aide de la commande :
  SHOW search_path;

- Pour ajouter un schéma au chemin, on écrit :
  SET search_path TO mon_schema , public;
  On peut aussi écrire
  SET search_path TO mon_schema;

  Dans ce cas, le schéma public n'est plus accessible sans qualification explicite

- Pour afficher les schémas existants:
  \dn

- Pour déplacer une table vers un autre schéma:
  Alter table nomtable set schema nomSchema

Schémas

Exemple sur les schémas

Supposons que dans une base de données ***Maboutique*** on a créé 2 schémas ***Mamoudzou*** et **Kaweni** qui portent chacun les mêmes noms de tables ***employes*** et ***ventes***.

La requete « *Select * from ventes; »* pose un ambigu car on ne sait pas si la table ventes en question est celle du schéma Mamoudzou ou celle de Kaweni.

Par conséquent, le nom du schéma doit être qualifié:

Pour Mamoudzou : *Select * from mamoudzou.ventes;*

Et pour Kaweni: *Select * from kaweni.ventes;*

Nous pouvons maintenant combiner ces requêtes avec des opérateurs tels que **UNION** et **UNION ALL** pour obtenir quelque chose de plus près à ce que le chef de la direction veut:

```
select 'mamoudzou', sum(montant) from mamoudzou.ventes  where annee=2013
UNION ALL
select 'kaweni', sum(montant) from kaweni.ventes  where  annee=2013;
```

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Sauvegardes et restaurations

Sauvegardes – pg_dump

Le principe est de produire un fichier texte de commandes SQL appelé «fichier dump », qui, si on le renvoie au serveur, recrée une base de données identique à celle sauvegardée

L'usage basique est :

pg_dump –U nomUtilisateur [options] *base_de_donnees > fichier_de_sortie*

Options: on a plusieurs options mais on donne ici 2 options les plus utilisées:

-t nomtable : sélectionne seulement la table indiquée:

-s : sauvegarde seulement les definitions des objets, et pas les données contenues.

Exemple :

Sauvegarder la base de données **maBoutique** sous le nom **maBoutique_dump.sql** dans le répertoire **C:\TEST :**

pg_dump -U postgres maBoutique > c:\TEST\maBoutique_dump.sql

Sauvegardes et restaurations

Restaurations

Les fichiers texte créés par pg_dump peuvent être lus par le programme psql. La syntaxe générale d'une commande de restauration

est :

Psql –U nomUtilisateur base_de_donnees < fichier_d_entree

où *fichier_d_entree est le fichier en sortie de la commande pg_dump. La base de données base_de_donnees n'est* pas créée par cette commande. Elle doit être créée avant d'exécuter psql.

Exemple*:*

Psql –U postgres maboutique1 < c:\TEST\maboutiquedump.sql

(maboutique1 est déjà créée avant le lancement de la commande)

Sauvegardes et restaurations

Utilisation de pg_dumpall

Pg_dump ne sauvegarde qu'une seule base à la fois, et ne sauvegarde pas les informations relatives aux rôles . Pour avoir une sauvegarde de tout le contenu d'un cluster, on utilise la commande **pg_dumpall** dont l'utilisation basique est :

Pg_dumpall -U nomUtilisateur > *fichier_de_sortie*

Pour la restauration on utilise psql :

psql -f *fichier_d_entree postgres*

Sauvegarde automatique de BDD en Windows

Nous voulons sauvegarder la base de données **maBasedeDonnee** automatiquement dans le repertoire « C:\SauvegardePostgreSql »

Nous nous connectons au serveur de base de données avec le login **postgres** et le mot de passe **monpass**.

Pour se faire, mettre dans une commande batch (.bat) les différentes lignes de commande ci-dessous (par exemple c:\sauvegardePostgreSql.bat):

```
SET JOUR=%date:~-10,2%
SET ANNEE=%date:~-4%
SET MOIS=%date:~-7,2%
SET HEURE=%time:~0,2%
SET MINUTE=%time:~3,2%
SET SECOND=%time:~-5,2%
SET REPERTOIR=C:\SauvegardePostgreSql
SET FICHIER=%REPERTOIR%\Sauvegarde_du_%JOUR%_%MOIS%_%ANNEE%_à_%HEURE%h_%MINUTE%mn.sql
IF NOT exist  %REPERTOIR%  md  %REPERTOIR%
SET PGPASSWORD=monpass
pg_dump –U nomUtilisateur maBasedeDonnee > %FICHIER%
```

Remarque:

Utiliser le Planificateur de taches Windows pour l'automatisation.

Un exemple de fichier resultat: **Sauvegarde_du_03_01_2014_A_16h_27mn.sql**

Sauvegarde automatique de BDD en Linux

Nous voulons sauvegarder la base de données **mabasededonne** automatiquement dans le repertoire « ~/SauvegardePostgreSql »
Nous nous connectons au serveur de base de données avec le login **postgres** et le mot de passe **monpass**.
Pour se faire, mettre dans un fichier bash(.sh) les différentes lignes de commande ci-dessous (par exemple ~/ SauvegardePostgreSql /sauvegardePostgreSql.sh):

```
#! /bin/bash
Bdd=mabasededonnee
Export PGPASSWORD=monpass
hm=$(date +%X)
h=${hm:0:2}
m=${hm:3:2}
fichier=${bdd }_$(date  +%A_%d-%m-%Y)_${h}_${m}mn.sql
pg_dump –U postgres  mabasededonnee  >  $fichier
```

Rendre le fichier bash en fichier executable:

```
chmod  +x  ~/ SauvegardePostgreSql /sauvegardePostgreSql.sh
```

Remarque:

Utiliser le Planificateur de taches **gnome-schedule** pour l'automatisation.

Un exemple de fichier resultat: **mabasededonnee_Lundi_16-06-2014_12h-19mn.sql**

Plan de formation

1. Base de données PostgreSql
2. Langage SQL
3. Système de règles
4. Héritage - Partitionnement
5. Schémas
6. Sauvegarde et Restauration
7. Gérer les utilisateurs et leurs droits

Gérer les utilisateurs et leurs droits

Créer un utilisateur en commande externe

Syntaxe:

createuser [options] [nom_utilisateur]

Description

createuser crée un nouvel utilisateur **PostgreSQL**

Gérer les utilisateurs et leurs droits

Créer un utilisateur – les options (1/4)

nom_utilisateur	Le nom de l'utilisateur à créer. Ce nom doit être différent de tout rôle de l'instance courante de PostgreSQL.
-c numéro, **--connection-limit=numéro**	Configure le nombre maximum de connexions simultanées pour le nouvel utilisateur. Par défaut, il n'y a pas de limite.
-d, --createdb	Le nouvel utilisateur est autorisé à créer des bases de données.
-D, --no-createdb	Le nouvel utilisateur n'est pas autorisé à créer des bases de données. Cela correspond au comportement par défaut.
-e, --echo	Les commandes engendrées par **createuser** et envoyées au serveur sont affichées.

Gérer les utilisateurs et leurs droits

Créer un utilisateur – les options (2/4)

--interactive	Demande le nom de l'utilisateur si aucun n'a été fourni sur la ligne de commande, et demande aussi les attributs équivalents aux options -d/-D, -r/-R, -s/-S si les options en ligne de commande n'ont pas été explicitement indiquées.
-l, --login	Le nouvel utilisateur est autorisé à se connecter (son nom peut être utilisé comme identifiant initial de session). Comportement par défaut.
-L, --no-login	Le nouvel utilisateur n'est pas autorisé à se connecter. (Un rôle sans droit de connexion est toujours utile pour gérer les droits de la base de données.)

Gérer les utilisateurs et leurs droits

Créer un utilisateur – les options (3/4)

-P, --pwprompt	L'utilisation de cette option impose à **createuser** d'afficher une invite pour la saisie du mot de passe du nouvel utilisateur.
-r, --createrole	Le nouvel utilisateur est autorisé à créer de nouveaux rôles (il possède le privilège CREATEROLE).
-R, --no-createrole	Le nouvel utilisateur n'est pas autorisé à créer de nouveaux rôles. Cela correspond au comportement par défaut.
-s, --superuser	Le nouvel utilisateur a les privilèges super utilisateur.
-S, --no-superuser	Le nouvel utilisateur n'a pas les privilèges super utilisateur. Cela correspond au comportement par défaut.

Gérer les utilisateurs et leurs droits

Créer un utilisateur – les options (4/4)

-h hôte, --host=hôte	Le nom de l'hôte sur lequel le serveur est en cours d'exécution.
-p port, --port=port	Le port TCP,
-U nomutilisateur, --username=nomutilisateur	Nom de l'utilisateur utilisé pour la connexion (pas celui à créer).
-w, --no-password	Ne demande jamais un mot de passe.
-W, --password	Force **createuser** à demander un mot de passe (pour la connexion au serveur, pas pour le mot de passe du nouvel utilisateur).

Gérer les utilisateurs et leurs droits

Exemples (1/2)

Créer un utilisateur joe sur le serveur de bases de données par défaut :

createuser joe

Pour créer un utilisateur joe avec le mode interactif :

createuser --interactive joe

Shall the new role be a superuser? (y/n) n

Shall the new role be allowed to create databases? (y/n) n

Shall the new role be allowed to create more new roles? (y/n) n

Créer le même utilisateur, joe, sur le serveur *eden*, *port 5000*, sans interaction, avec affichage de la commande sous-jacente :

createuser -h eden -p 5000 -S -D -R -e joe

CREATE ROLE joe NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT LOGIN;

Gérer les utilisateurs et leurs droits

Exemples (2/2)

Créer l'utilisateur joe, super utilisateur, et lui affecter immédiatement un mot de passe :

createuser -P -s -e joe

Enter password for new role: xyzzy

Enter it again: xyzzy

CREATE ROLE joe PASSWORD 'xyzzy' SUPERUSER CREATEDB CREATEROLE INHERIT LOGIN;

CREATE ROLE

Dans l'exemple ci-dessus, le nouveau mot de passe n'est pas affiché lorsqu'il est saisi. Il ne l'est ici que pour plus de clarté. Comme vous le voyez, le mot de passe est chiffré avant d'être envoyé au client. Si l'option *--unencrypted* est utilisé, le mot de passe *apparaîtra* dans la commande affichée , donc vous ne devez pas utiliser *-e* dans ce cas, surtout si quelqu'un d'autre voit votre écran à ce moment.

Gérer les utilisateurs et leurs droits

Supprimer un utilisateur (1/3)

dropuser [options] [nomutilisateur]

nomutilisateur	Le nom de l'utilisateur PostgreSQL à supprimer. Un nom est demandé s'il n'est pas fourni sur la ligne de commande et que l'option -i/--interactive est utilisé.
-e, --echo	Les commandes engendrées et envoyées au serveur par **dropuser** sont affichées.
-i, --interactive	Une confirmation est demandée avant la suppression effective de l'utilisateur.

Gérer les utilisateurs et leurs droits

Supprimer un utilisateur (2/3)

-h hôte, --host=hôte	Le nom d'hôte de la machine sur lequel le serveur fonctionne
-p port, --port=port	Le port TCP ou l'extension du fichier du socket local de domaine Unix sur lequel le serveur attend les connexions.
-U nomutilisateur, --username=nomutilisateur	Le nom de l'utilisateur utilisé pour la connexion.
-w, --no-password	Ne demande jamais un mot de passe.
-W, --password	Force **dropuser** à demander un mot de passe avant la connexion à une base de données.

Gérer les utilisateurs et leurs droits

Supprimer un utilisateur (3/3) – Exemples

Supprimer l'utilisateur ***joe*** de la base de données par défaut :

dropuser joe

Supprimer l'utilisateur ***joe*** sur le serveur hébergé sur l'hôte ***eden***, qui écoute sur le ***port 5000***, avec demande de confirmation et affichage de la commande sous-jacente :

dropuser -p 5000 -h eden -i -e joe

Role "joe " will be permanently removed.Are you sure? (y/n)

y

DROP ROLE joe;

Gérer les utilisateurs et leurs droits

Création d'utilisateurs en mode console au serveur

CREATE USER nom [[WITH] option [...]]

où option peut être :

CREATEDB | NOCREATEDB |
CREATEUSER | NOCREATEUSER |
IN GROUP nomgroupe [, ...] |
[ENCRYPTED | UNENCRYPTED] PASSWORD 'mot_de_passe' |
VALID UNTIL 'temps_absolu'

Exemples

Créez un utilisateur sans mot de passe :
CREATE USER jonathan;
Créez un utilisateur avec un mot de passe :
CREATE USER davide WITH PASSWORD 'jw8soF4';
Créez un utilisateur avec un mot de passe qui est valide jusqu'à la fin 2004
CREATE USER miriam WITH PASSWORD 'jw8soF4' VALID UNTIL '2005-01-01';
Créez un compte où l'utilisateur peut créer des bases de données :
CREATE USER manuel WITH PASSWORD 'jw8soF4' CREATEDB;

Notes

Utilisez *ALTER USER* pour changer les attributs d'un utilisateur, et *DROP USER* pour le supprimer.

Gérer les utilisateurs et leurs droits

Definir des droits d'accès

La définition des droits d'accès se fait par **GRANT**

Syntaxes:

GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES } | ALL [PRIVILEGES] } ON {nom_table [, ...] | ALL TABLES IN SCHEMA nom_schéma [, ...] } TO {nom_user|PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (nom_colonne [, ...]) [, ...] | ALL [PRIVILEGES] (nom_colonne [, ...]) } ON nom_table [, ...] TO {nom_user | PUBLIC } [, ...] [WITH GRANT OPTION]

Gérer les utilisateurs et leurs droits

Les droits possibles:

SELECT , INSERT, UPDATE, DELETE :

- Autorise respectivement la selection , l'insertion , la mise à jour et la suppression dans une table

REFERENCES:

- Ce privilège est requis sur la table de référence et la table qui référence pour créer une contrainte de clé étrangère.

CREATE :

- Pour les bases de données, autorise la création de nouveaux schémas dans la base de données.
- Pour les schémas, autorise la création de nouveaux objets dans le schéma.

USAGE :

- Pour les *schémas*, autorise l'accès aux objets contenus dans le schéma indiqué
- Pour les *séquences*, ce droit autorise l'utilisation des séquences dans les tables.

Gérer les utilisateurs et leurs droits

GRANT - Exemples

Donner le droit d'insertion à tous les utilisateurs sur la table films :

GRANT INSERT ON films TO PUBLIC;

Donner tous les droits à l'utilisateur ***manuel*** sur la vue ***genres*** :

GRANT ALL PRIVILEGES ON genres TO manuel;

Gérer les utilisateurs et leurs droits

GRANT - Exemples

Autres Exemples:

GRANT SELECT ON matable TO PUBLIC;

GRANT SELECT, UPDATE, INSERT ON matable TO admin;

GRANT SELECT (col1), UPDATE (col1) ON matable TO miriam;

GRANT USAGE ON SCHEMA vente to secretariat;

GRANT USAGE ON SEQUENCE commande_id_seq to secretariat;

\dp

permet d'obtenir des informations sur les droits existants pour les tables et colonnes.

Gérer les utilisateurs et leurs droits

REVOKE

REVOKE — supprime les droits d'accès

Exemples:

Enlever au groupe ***ChefDepot*** le droit de modifier le prix d'un article :

REVOKE UPDATE(prix) ON article FROM ChefDepot;

Enlever au groupe ***public*** le droit d'insérer des lignes dans la table films :

REVOKE INSERT ON films FROM PUBLIC;

Supprimer tous les droits de l'utilisateur ***manuel*** sur la vue genres :

REVOKE ALL PRIVILEGES ON genres FROM manuel;

Supprimer l'appartenance de l'utilisateur ***joe*** au rôle ***admins*** :

REVOKE admins FROM joe;

Gérer les utilisateurs et leurs droits

Afficher les droits sur les tables

\z affiche la liste des permissions d'accès sur les différentes tables.

⇒ *\z matable*

```
                    Access privileges for database "lusitania"
Schema |  Name   | Type  |                  Access privileges
--------+---------+-------+-----------------------------------------------------------
Public  | matable  | table | {miriam=arwdxt/miriam,=r/miriam,"group todos=arw/miriam"}(1 row)
```

r -- SELECT ("lecture")
w -- UPDATE ("écriture")
a -- INSERT ("ajout")
d – DELETE
x – REFERENCES
t -- TRIGGER
X -- EXECUTE
U – USAGE
C – CREATE

Pour afficher les différents utilisateurs et groupes:

\du

Gérer les utilisateurs et leurs droits

Création de rôles en mode console au serveur

CREATE ROLE nom [[WITH] option [...]]

où *option* peut être :

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| IN ROLE *nom_role* [, ...]
| IN GROUP *nom_role* [, ...]
| ROLE *nom_role* [, ...]
| ADMIN *nom_role* [, ...]
| USER *nom_role* [, ...]

CREATE ROLE ajoute un nouveau rôle dans une grappe (cluster) de bases de données .

Gérer les utilisateurs et leurs droits

Création de rôles - Exemples

Créer un simple rôle DIRECTION:

CREATE ROLE DIRECTION ;

Créer un rôle qui peut créer des bases de données et gérer des rôles :

CREATE ROLE ADMIN WITH CREATEDB CREATEROLE;

Gérer les utilisateurs et leurs droits

SET ROLE

SET ROLE initialise le role de la session en cours .

Syntaxe:

SET [SESSION | LOCAL] ROLE nom_rôle
SET [SESSION | LOCAL] ROLE NONE
RESET ROLE

Les formes ***NONE*** et ***RESET*** annule le role en cours.

Exemples

```
SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
 ------------- +-------------
peter          | peter
```

```
SET ROLE  'paul';
```

```
SELECT SESSION_USER, CURRENT_USER;
session_user  | current_user
 -----------  --+-------------
     peter        | paul
```

Gérer les utilisateurs et leurs droits

ALTER ROLE

ALTER ROLE modifie un rôle de base de données.

Syntaxes:

ALTER ROLE nom [[WITH] option [...]]

où *option* peut être :

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT *limiteconnexion*
| [ENCRYPTED | UNENCRYPTED] PASSWORD '*motdepasse*'
| VALID UNTIL '*dateheure*'

ALTER ROLE nom RENAME TO nouveau_nom

Gérer les utilisateurs et leurs droits

ALTER ROLE - Exemples

Donner à un rôle la capacité de créer d'autres rôles et de nouvelles bases de données :

ALTER ROLE DIRECTION CREATEROLE CREATEDB;

Cas pratique sur la creation des roles et des droits (1/2)

Créer un utilisateur superviseur ***admin :***

CREATE USER admin SUPERUSER password 'p@admin';

Créer les groupes ***secretariat, admin_secretariat,admin_imie***:

CREATE ROLE secreetariat;

CREATE ROLE admin_secretariat;

CREATE ROLE admin_imie;

Créer les utilisateurs ***imie, anna, roseline:***

CREATE USER imie password 'p@imie' ;

CREATE USER anna password 'p@anna';

CREATE USER roseline password 'p@roseline';

Rattacher au groupe ***secretariat*** les utilisateurs ***anna, roseline***:

GRANT ***secretariat to anna;***

GRANT ***secretariat to roseline;***

ou

CREATE USER marie in role secretaire password 'p@marie';

Rattacher au groupe ***admin_secretariat*** l'utilisateur ***anna***:

GRANT admin_***secretariat to anna;***

Cas pratique sur la creation des roles et des droits (2/2)

Créer une base de donnée de nom ***imie*** de propriétaire ***imie***:

CREATE DATABASE imie OWNER imie;

Ou , si la base de donnée ***imie*** est déjà créée***:***

ALTER DATABASE imie owner to imie;

---(A cet effet l'utilisateur imie a plein droit sur la base de donnée imie)---

Se connecter à la base de donnée imie avec l'utilisateur ***imie:***

\c imie imie;

Créer la table commande:

CREATE TABLE commande(id serial primary key, datecommande date, client_id);

Donner les droits ***SELECT,UPDATE,DELETE,INSERT*** au groupe ***secretariat*** sur la table ***commande***:

GRANT ***SELECT,UPDATE,DELETE,INSERT ON TABLE commande to secretariat;***

Donner le droit ***DELETE*** au groupe ***admin_secretariat*** sur la table ***commande***:

GRANT ***DELETE ON TABLE commande to admin_secretariat;***

Enlever le droit ***DELETE*** au groupe ***secretaria t*** sur la table ***commande***:

REVOKE ***DELETE ON TABLE commande from secretariat;***

Connecter à la base de donnée ***imie*** à partir de l'invite commande (commande shell) :

>psql –U imie

Le ficher pg_hba.conf (1/2)

L'authentification du client est contrôlée par un fichier nommé ***pg_hba.conf*** et situé par défaut dans le répertoire *data pour windows* et dans *etc/postgresql/9.4/main* par exemple pour Linux

Un enregistrement peut avoir l'un des formats suivants:

TYPE	DATABASE	USER	ADDRESS	METHOD

Type: Indique le type de connexion. Par exemple:

host intercepte les tentatives de connexion par TCP/IP

local intercepte les tentatives de connexion Linux

hostssl intercepte les seules tentatives de connexions par TCP/IP qui utilisent le chiffrement SSL.

hostnossl n'intercepte que les tentatives de connexion qui n'utilisent pas SSL. Il a une logique opposée à *hostssl* :

Database Indique les noms des bases de données concernées par l'enregistrement. Une valeur ***all*** indique qu'il concerne toutes les bases de données. Des noms de bases de données multiples peuvent être fournis en les séparant par des virgules. Un fichier contenant des noms de bases de données peut être indiqué en faisant précéder le nom du fichier de @.

User Indique les utilisateurs de la base de données auxquels cet enregistrement correspond. Plusieurs noms d'utilisateurs peuvent être fournis en les séparant par des virgules. Un fichier contenant des noms d'utilisateurs peut être indiqué en faisant précéder le nom du fichier de @.

Address Indique la plage d'adresses IP client à laquelle correspond cet enregistrement.

Le ficher pg_hba.conf (2/2)

Method Indique la méthode d'authentification à utiliser lors d'une connexion via cet enregistrement. Les choix peuvent être:

Trust Autorise la connexion sans condition.

Reject Rejette la connexion sans condition. Ce cas est utile pour « filtrer » certains hôtes d'un groupe

Md5 Demande au client de fournir un mot de passe chiffré MD5 pour l'authentification

Password Requiert que le client fournisse un mot de passe non chiffré pour l'authentification

Ident Récupère le nom de l'utilisateur du système d'exploitation du client

Exemple :

```
# Permettre à n'importe quel utilisateur du système local de se connecter
# à la base de données sous n'importe quel nom d'utilisateur
# en utilisant les connexions TCP/IP locales.
#
# TYPE  DATABASE    USER        ADDRESS             METHOD
host    all         all       127.0.0.1/32        trust
```

Sources

- Livres :
  - *PostgreSQL Administration et Exploitation par Sebastien LARDIERE*
  - *Utiliser PostgreSQL par Domonique COLOMBANI*

- Sites :
  - *http:// www.developpez.com*
  - *http://docs.postgresql.fr*
  - *http://www.commentcamarche.net*
  - *http://sql.sh/*