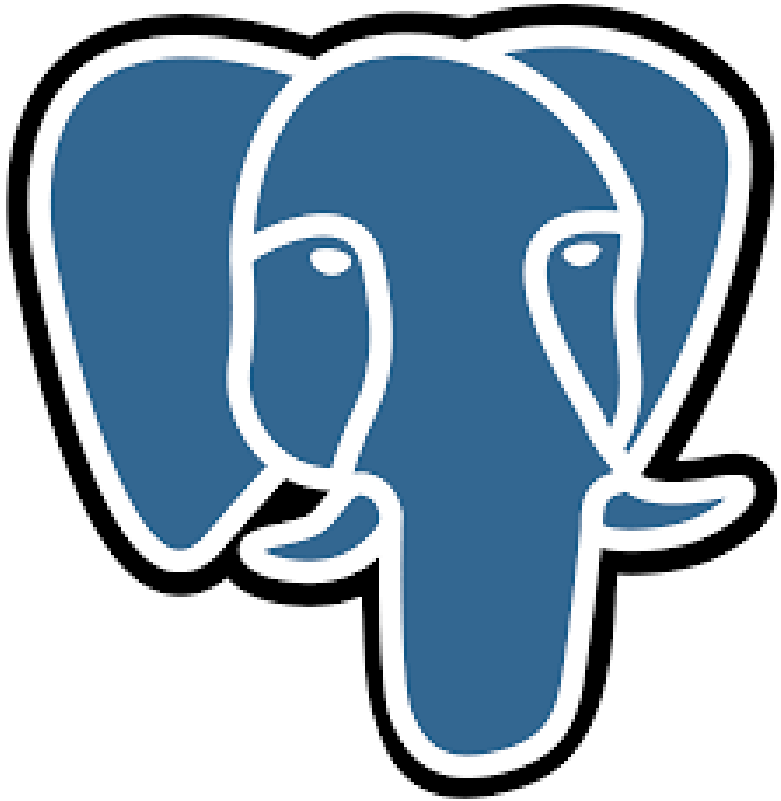
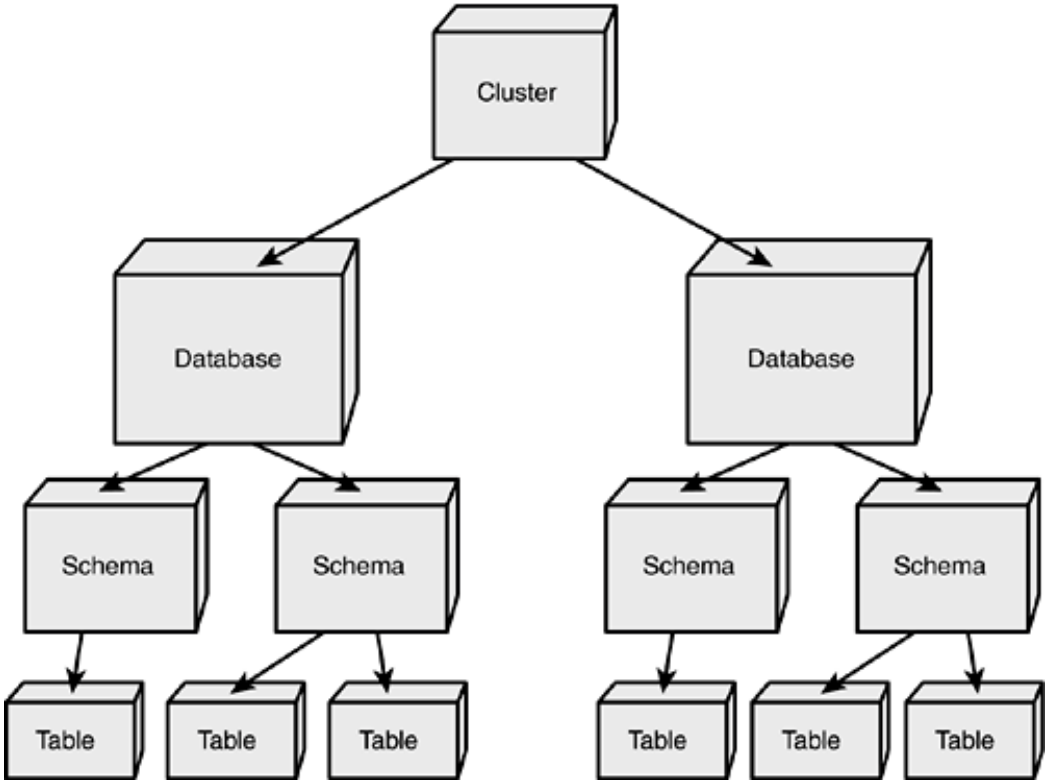
Postgresql – aspects administratifs

Programme

- Utilisateurs / groupes
- Authentification
- Privilèges
- Schémas
- Réplication Master – Slave



Repère initial : des boites dans des boites dans des boites



Une analogie

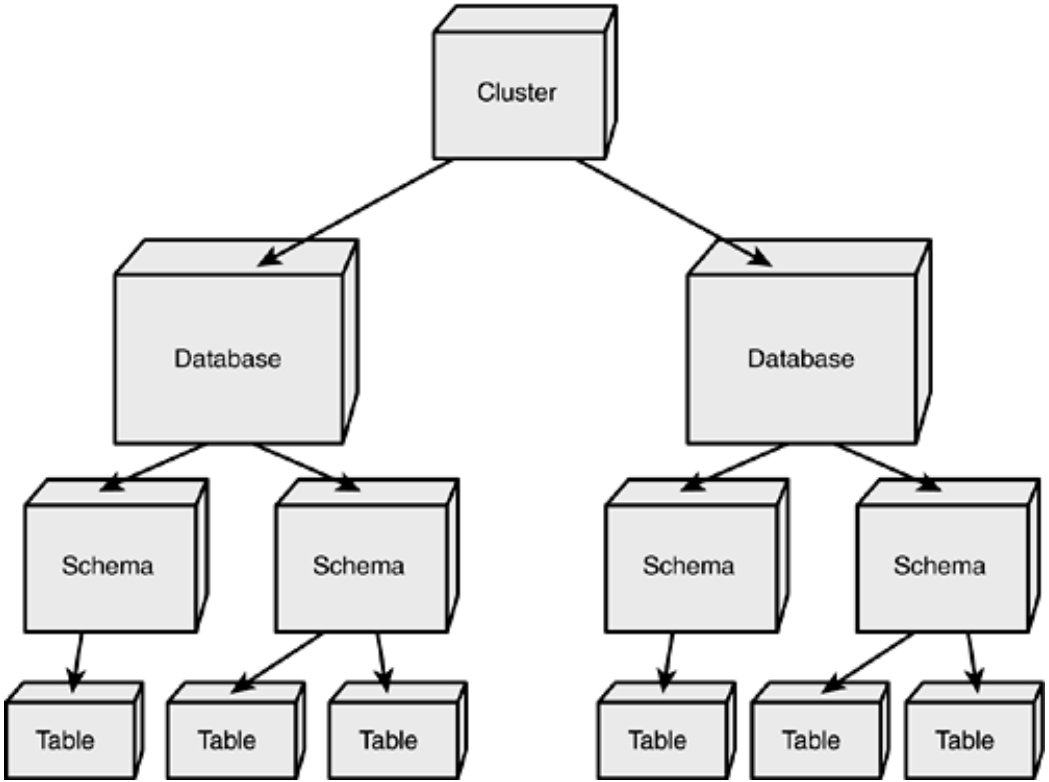
cluster : un disque dur (stockage de données, espace)

database : une partition (authentification, séparation)

schema : un dossier (organisation, rangement, séparation par type de données)

table : un fichier de données (organisation des données elles mêmes)

Repère initial : des boites dans des boites dans des boites



L'objectif de ce cours est de comprendre :

- « *ou vivent* » les **users**, **procédures** stockées et autres objets rencontrés précédemment
- Comment générer finement et efficacement l'accès au données par la configuration des **niveaux intermédiaires**
- Héritage des droits

Utilisateurs et groupes

Créons :

- Une nouvelle base
- Un nouvel utilisateur

```
CREATE DATABASE test_db;
CREATE USER test_user WITH PASSWORD 'test';
```

Remarquez les faits suivants :

- Par défaut, une base de données appartient à son créateur (`\l`)
- Par défaut, un utilisateur ne dispose d'aucun **attribut** et n'appartient à aucun **groupe** (`\du`)
- Sauf précision du contraire, une base de données est accessible à n'importe quel utilisateur (essayez de vous connecter à test_db comme test_user pour vous en assurer) mais non ses données

Utilisateurs et groupes

Comment se fait-il que test_user puisse :

- Se connecter à test_db
- Lister ses tables
- Mais non à interagir avec le contenu de ces dernières ?

Ces trois niveaux tiennent à des composants différents

- Privilèges de **connexions**
- Accès à un **schéma** et son **search path** associé
- Privilège sur les tables

Privilèges de connexions

Restriction de l’accès à la DB

```
REVOKE connect ON DATABASE test_db FROM PUBLIC;
```

Notez que test_user ne peut plus se connecter à la base

Donner les privilèges de connexion à un utilisateur

```
GRANT connect ON DATABASE test_db TO test_user;
```

Les lui enlever

```
REVOKE connect ON DATABASE test_db FROM test_user;
```

Groupes et droits

Sachant que nous allons devoir créer pas mal d'utilisateurs, il serait bon que nous n'ayons pas besoin de gérer les droits individuellement

Un peu comme sur UNIX, les droits peuvent être gérés au niveau d'un **groupe** et s'appliquer à l'ensemble de ses membres

Nous allons

- Créer un groupe : **CREATE GROUP test_group;**
- Lui donner les droits de connexion :

GRANT connect ON DATABASE test_db TO test_group;

- Lui ajouter l'utilisateur test_user :

ALTER GROUP test_group ADD USER test_user;

Groupes et droits

Nota Bene :

- l’appartenance à un groupe peut être énoncée au moment du **CREATE USER**
- Un même utilisateur peut appartenir à **plusieurs groupes**

A vous : à l’aide des informations précédentes, créer

- Un groupe **admin** et un groupe **web**
- 2 utilisateurs capables de se connecter à la test_db
  – test_admin (admin) : a le droit de créer des utilisateurs mais pas de db
  – test_web_app (web)

Schémas

Pour les besoins de la démo qui va venir, vous allez

- créer une table dans le test_db et
- Y écrire quelques lignes

```
CREATE TABLE test_table(
    id SERIAL,
    first_name VARCHAR(80),
    last_name VARCHAR(80));
```

Si maintenant vous \d

Vous devez voir que test_table a été créée dans un **schéma** du nom de public

Qu’est-ce que cela implique ?

Schémas : observations préliminaires

- Lorsque vous exécutez **SELECT * FROM test_table** en réalité, vous exécutez **SELECT * FROM public.test_table;**
- **Pourquoi ?** Allez regarder **SHOW search_path;**
- **\l** permet à **test_user** de lister les tables pour cette même raison
- Exécuter une seconde fois **CREATE TABLE test_table()** vous renvoie une erreur ? Normal, ce nom est déjà pris

Conclusions préliminaires :

- un schéma est un genre de PATH
- Sauf précision du contraire tout est créé dans public (qui existe par défaut)

Schéma : création

Nous allons créer un nouveau schéma

```
CREATE SCHEMA private;
```

Créez la même table que précédemment mais dans ce nouveau schéma

```
CREATE TABLE private.test_table(
    id SERIAL,
    first_name VARCHAR(80),
    last_name VARCHAR(80));
```

Notez qu'aucune erreur se produit : il n'y a pas de conflit de namespace

Schéma : search path

Comment se fait-il que nous n'ayons pas eu à préciser les **noms qualifiés** (e.g. **public.ma_table**) jusqu'à présent

Cela tient au fait que chaque utilisateur dispose d'un **search path**

SHOW search_path;

Changer le search path d'un utilisateur

ALTER ROLE username SET search_path = schema1,schema2,schema3,etc;

Pendant que nous y sommes, assurons nous que personne d'autre ne puisse créer de nouveaux objets dans le schéma public

REVOKE CREATE ON SCHEMA public FROM PUBLIC;

Schéma : search path

A l’aide des informations précédentes, modifiez le **search path** des utilisateurs de la manière suivante :

- test_user_1 : public et private
- test_user : private
- test_web_app : public

Question : après cette édition, test_web_app peut-il quand même afficher la liste des tables du schéma private ?

Moralité : les schémas sont un moyen commode i) d’organiser ii) en évitant des conflits de noms mais il ne se substituent pas à une vrai gestion des droits de lecture

Schéma : en résumé usages typiques des schémas

- Pour autoriser **beaucoup d'utilisateurs** d'utiliser une base de données sans se gêner les uns les autres
- Pour organiser des objets de bases de données en **groupes logiques** afin de faciliter leur gestion.
- Les **applications tierces** peuvent être mises dans des schémas séparés pour qu'il n'y ait pas de collision avec les noms d'autres objets.

Tables et privilèges

Hormis le fait d'en être propriétaire, un utilisateur peut interagir avec une table pour peut qu'il dispose de **privilèges** sur cette dernière. On peut choisir de donner :

- Tous les privilèges : GRANT ALL ON table TO user;
- Seulement l'usage de certains privilèges

  SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE et ALL PRIVILEGES

Tables et privilèges : un grand pouvoir implique de grandes responsabilités

Sur la base des utilisateurs et groupes créés précédemment, donner les droits suivants sur la table **test_table**

- Tous les droits à l'admin
- SELECT, INSERT, UPDATE à test_user
- SELECT a test_web_app

Une vieille technique pour protéger les données d'une base consiste à ne jamais donnée le privilège DELETE mais confier plutôt la suppression des lignes à une **procédure stockée (**exécutée avec les privilèges de celui qui l'aura définie**)**

Faisons ce qu'il faut pour que test_user puisse supprimer une ligne par id

Tables et privilèges : un grand pouvoir implique de grandes responsabilités

A la différence d'une **fonction**, une **procédure** (introduit en postgresql>11) ne renvoi rien (elle exécute simplement)

```
CREATE OR REPLACE PROCEDURE
remove_data(a integer)
LANGUAGE SQL
AS $$
DELETE FROM test_table WHERE id = a ;
$$
SECURITY DEFINER;
```

Comme test_user, appelé la fonction pour supprimer la ligne 1 de la table

```
CALL remove_data(1);
```

Un problème ? test_user n'a peut-être pas le droit d'utiliser la fonction

- Résolvez ce problème
- Modifier la procédure de manière à ce qu'elle puisse supprimer la ligne d'une table quelconque (passée en argument)

Tables et privilèges : un grand pouvoir implique de grandes responsabilités

Avantages de cette méthode :

- Plus de DELETE accidentel ou hasardeux
- Input normalisé (e.g. passer l'id de la ligne à supprimer)
- Données intègres en cas d'injection SQL
- **Révocation des droits centralisée :** priver l'utilisateur de l'**usage d'une procédure** stockée VS **changer ses privilèges** sur l'ensemble des tables de la base de données

Réplication de Base de données : Fonction et mise en place

Deux grands usages

Failover :

- En cas d’indisponibilité du serveur principal, le serveur secondaire prend le relais

Sauvegardes :

- En cas de faille majeur du système de fichier, une version dupliquée de la base demeure

Réplication

Master-Master

chaque serveur peut recevoir une **requête d'écriture**, auquel cas il met à jour sa base puis se synchronise avec l'autre serveur

Master-Slave

seul le **serveur maître** peut recevoir une **requête d'écriture**

Nous allons mettre en place une réplication master slave. Prérequis :

- Deux instances de Postgresql
- Même architecture matérielle

Réplication Master Slave : configuration commune

postgresql.conf sur **Master** et **Slave**

wal_level = hot_standby

max_wal_senders = 1

wal_keep_segments = 256

checkpoint_segments = 8

hot_standby = on

Explication

- **Wal Write-Ahead Log**
- **Hot standby :** permettra au Slave de répondre aux requêtes en lecture seule

Réplication Master Slave : côté slave

c’est la présence du fichier recovery.conf qui fait que Postgresql traite le serveur comme une réplique