

Le langage de définition des données s'appuie sur trois ordres : `CREATE`, `ALTER` et `DROP`, déclinés pour chaque objet de la base de données. Chaque objet, appelé aussi relation, peut correspondre soit à un objet physique, et donc à des fichiers dans les systèmes de fichiers, soit à un objet logique, étant donc seulement une définition stockée dans les paramètres des bases de données.

A. Les espaces 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, PostgreSQL dispose d'un espace de tables, situé dans le répertoire du groupe de bases de données. Il est possible de créer d'autres espaces de tables, permettant ainsi à l'administrateur de choisir l'emplacement du stockage d'une table ou d'un index.

Plusieurs motifs peuvent amener un administrateur à créer un espace de tables :

- Le disque dur ou la partition ne disposent plus de suffisamment d'espace libre. Les espaces de tables permettent dans ce cas d'utiliser plusieurs disques durs différents pour un même groupe de base de données, sans utiliser de systèmes de volumes logiques, comme LVM dans les systèmes GNU/Linux.
- L'utilisation des tables et des index provoque une saturation des écritures et des lectures du disque dur où est situé l'espace de tables par défaut. Même sur des systèmes performants, utilisant par exemple des bus SCSI, la montée en charge d'une application peut amener l'administrateur à créer d'autres espaces de tables, sur des disques durs différents. Ainsi, les écritures et les lectures de fichiers de tables et d'index sont réparties sur plusieurs supports physiques, améliorant les performances.

Un espace de tables est donc un outil, utilisable par l'administrateur du serveur de bases de données, permettant d'intervenir sur l'emplacement physique du stockage des tables et des index. L'administrateur peut donc choisir, table par table et index par index, quelle que soit la base de données, l'emplacement d'un fichier, optimisant ainsi les volumes utilisés, les écritures et les lectures.

Un espace de tables n'est pas spécifique à une base de données. Il fait partie d'un groupe de bases de données, et est, par défaut, utilisable depuis toutes les bases de données. Une gestion fine des permissions sur les espaces de tables permet à l'administrateur de maîtriser la répartition des fichiers, en fonction des bases de données et des rôles utilisés.

La première étape, avant d'initialiser l'espace de tables depuis PostgreSQL, est de créer un répertoire pour cet usage. Selon les besoins, il peut s'agir d'un répertoire sur un nouveau disque dur ou sur une partition d'un disque dur déjà présent. Les étapes suivantes permettent d'initialiser ce répertoire, et de positionner les droits nécessaires pour que l'utilisateur `postgres` soit le propriétaire :

```
[root]# mkdir -p /data2/postgres/tblspc2/
[root]# chown postgres:postgres /data2/postgres/tblspc2/
[root]# chmod 700 /data2/postgres/tblspc2/
```

Si le répertoire existe déjà, il doit être vide, et appartenir à l'utilisateur du système d'exploitation qui exécute l'instance de PostgreSQL.

Une fois le répertoire créé, il suffit de l'enregistrer dans l'instance de PostgreSQL, en lui donnant un nom qui permet de l'identifier. Le synopsis de la commande est le suivant :

```
postgres=# CREATE TABLESPACE nomtblspc [ OWNER nomrole ]
LOCATION 'repertoire';
```

Par défaut, l'espace de tables ainsi créé appartient à l'utilisateur qui exécute la commande. Seul un super-utilisateur peut créer un espace de tables, mais il peut transmettre l'appartenance à un autre utilisateur par l'intermédiaire de l'option `OWNER`.

Une fois que l'espace de tables existe, il peut être utilisé au moment de la création ou de la modification des tables et des index.

1. Modification d'un espace de tables

Il est possible de modifier un espace de tables existant. Deux paramètres sont modifiables : le nom et le propriétaire. La commande `ALTER TABLESPACE` permet de faire ces deux modifications, comme dans les synopsis suivants :

```
postgres=# ALTER TABLESPACE nomtblspc1 RENAME TO nomtblspc2 ;
```

```
postgres=# ALTER TABLESPACE nomtblspc1 OWNER TO nomrole ;
```

2. Suppression d'un espace de tables

La suppression d'un espace de tables est possible s'il n'existe plus aucune table ni aucun index dans cet espace de tables.

Avant de supprimer l'espace de tables, il faut avoir déplacé dans un autre espace de tables tous les objets contenus, y compris ceux qui n'appartiennent pas à la base de données courante. Une fois que l'espace de tables est vide, l'ordre `DROP TABLESPACE` permet cette suppression, comme le montre le synopsis suivant :

```
postgres=# DROP TABLESPACE [ IF EXISTS ] nomtblspc ;
```

Une fois que l'espace de tables est supprimé dans l'instance de PostgreSQL, le répertoire du système de fichiers n'est plus utile et peut à son tour être supprimé.

L'option `IF EXISTS`, apparue dans la version 8.2, permet de ne pas provoquer d'erreur lors de la suppression si l'espace de table n'existe pas.

B. Les bases de données

Dans une instance de PostgreSQL, une base de données est un conteneur. Elle contient des schémas, et indirectement des tables, des index, et tous les objets utiles à une application. Elle accueille aussi les connexions depuis les applications clientes. En effet, lorsqu'une connexion est ouverte sur une base de données particulière, il n'est pas possible d'utiliser des objets créés dans d'autres bases de données. Il est donc important de répartir correctement les applications dans les bases de données, notamment en utilisant la notion de schéma. La création d'une base de données peut s'effectuer avec l'ordre `CREATE DATABASE`, ou avec la commande du système d'exploitation `createdb`.

Quelques paramètres permettent de personnaliser la création d'une base de données :

- Pour créer une base de données, il est nécessaire d'être super-utilisateur, ou d'avoir le privilège `CREATEDB`. En revanche, il est possible de transmettre l'appartenance à un utilisateur non-privilegié, avec l'option `OWNER`. L'option de la commande `createdb` est `-O` ou `--owner`.
- La base de données `template1` sert de modèle par défaut pour la création d'une autre base de données. Pour chaque base créée avec ce modèle, une copie de `template1` est faite, de telle sorte que tous les objets créés dans `template1` sont dupliqués dans la nouvelle base. La base `template0` fonctionne de la même façon, à la différence qu'il n'est pas possible de placer des objets dans cette base-modèle : `template0` reste une base vierge. Il est possible de choisir la base de données servant de modèle avec l'option `TEMPLATE`. Il est bien sûr possible de choisir n'importe quelle base de données existante. L'option de la commande `createdb` est `-T` ou `--template`.

- Un paramètre important d'une base de données est le choix du jeu de caractères. Ce paramètre détermine la façon dont les données seront stockées dans les tables et les index. Le jeu de caractères par défaut est déterminé à la création du groupe de base de données, mais il est possible de préciser un autre jeu de caractères avec l'option `ENCODING`. Il n'est pas possible de modifier cette option une fois la base de données créée. L'option de la commande `createdb` est `-E` ou `--encoding`.
- L'espace de tables par défaut est celui qui a été créé à l'initialisation du groupe de bases de données. Il est possible de créer d'autres espaces de tables et de l'associer avec une base de données, avec l'option `TABLESPACE`. Mais ce choix n'est qu'un paramètre par défaut pour la création des tables et des index, qui peut ne pas être suivi. L'option de la commande `createdb` est `-D` ou `-location`.

Le dernier paramètre, `CONNECTION LIMIT`, permet de contrôler le nombre de connexions entrantes qui, par défaut, n'est pas limité.

Le synopsis suivant montre l'ordre SQL permettant de créer une base de données :

```
postgres=# CREATE DATABASE nom [ [ WITH ] [ OWNER [=] role ]
[ TEMPLATE [=] modele ] [ ENCODING [=] codage ]
[ TABLESPACE [=] espace_table ] [ CONNECTION LIMIT
[=] limite_connexion ] ]
```

La commande ci-dessous permet de créer une base de données, appelée `clients`, avec le jeu de caractères UTF8 :

```
postgres=# CREATE DATABASE clients ENCODING UTF8 ;
```

Le synopsis suivant montre la commande du système d'exploitation :

```
[postgres]# createdb [option...] [nombase] [description]
```

La création de la base `clients` peut donc s'écrire comme ceci :

```
[postgres]# createdb -E UTF8 clients
```

Il est possible avec cette commande `createdb` de se connecter à un serveur distant, en utilisant les mêmes options que la commande `psql`. Par exemple, la commande suivante crée une base de données sur un serveur PostgreSQL distant :

```
[root]# createdb -E UTF8 -h 192.168.0.3 -p 5432 -U
postgres clients
```

➤ Une fois que la base de données est créée, il est possible de s'y connecter, depuis un client comme **psql**, **PgAdminIII** ou **phpPgAdmin**.

1. Modification d'une base de données

Une base de données peut être modifiée, une fois créée, avec l'ordre `ALTER DATABASE`. Plusieurs paramètres peuvent être modifiés, dont le nom, le propriétaire et la limite de connexions.

Les synopsis suivants montrent l'utilisation de l'ordre `ALTER DATABASE` :

```
postgres=# ALTER DATABASE nom CONNECTION LIMIT
limiteconnexion
```

```
postgres=# ALTER DATABASE nom RENAME TO nouveaunom
```

```
postgres=# ALTER DATABASE nom OWNER TO rôle
```

D'autres paramètres peuvent être ajoutés, comme par exemple certaines variables de configurations. Ces variables sont normalement associées au fichier de configuration de l'instance, mais peuvent être associées à la session, avec une autre valeur pour la base de données. La syntaxe suivante permet d'enregistrer ces variables :

```
postgres=# ALTER DATABASE nom SET paramètre { TO | = }
{ valeur | DEFAULT }
```

```
postgres=# ALTER DATABASE nom RESET paramètre
```