Module V0

Partitionnement déclaratif (introduction)

24.04

Table des matières

Sur ce document . . . 1
Chers lectrices & lecteurs, . . . 1
À propos de DALIBO . . . 1
Remerciements . . . 2
Forme de ce manuel . . . 2
Licence Creative Commons CC-BY-NC-SA . . . 2
Marques déposées . . . 3
Versions de PostgreSQL couvertes . . . 3

1/ Partitionnement déclaratif (introduction) 5
1.1 Principe & intérêts du partitionnement . . . 6
1.2 Partitionnement déclaratif . . . 8
1.2.1 Partitionnement par liste . . . 9
1.2.2 Partitionnement par liste : implémentation . . . 10
1.2.3 Partitionnement par intervalle . . . 10
1.2.4 Partitionnement par intervalle : implémentation . . . 11
1.2.5 Partitionnement par hachage . . . 12
1.2.6 Partitionnement par hachage : implémentation . . . 12
1.3 Performances & partitionnement . . . 14
1.3.1 Attacher/détacher une partition . . . 16
1.3.2 Supprimer une partition . . . 16
1.3.3 Limitations principales du partitionnement déclaratif . . . 17
1.4 Conclusion . . . 18
1.5 Quiz . . . 19

Les formations Dalibo 21
Cursus des formations . . . 21
Les livres blancs . . . 22
Téléchargement gratuit . . . 22

Sur ce document

Formation	Module V0
Titre	Partitionnement déclaratif (introduction)
Révision	24.04
PDF	https://dali.bo/v0_pdf
EPUB	https://dali.bo/v0_epub
HTML	https://dali.bo/v0_html
Slides	https://dali.bo/v0_slides

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

Chers lectrices & lecteurs,

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

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

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

À propos de DALIBO

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

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

[1] mailto:formation@dalibo.com

Remerciements

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

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

Forme de ce manuel

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

Licence Creative Commons CC-BY-NC-SA

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

Marques déposées

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

Versions de PostgreSQL couvertes

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

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

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

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

1/ Partitionnement déclaratif (introduction)

- Le partitionnement déclaratif apparaît avec PostgreSQL 10
- Préférer PostgreSQL 13 ou plus récent
- Ne plus utiliser l'ancien partitionnement par héritage.

Ce module introduit le partitionnement déclaratif introduit avec PostgreSQL 10, et amélioré dans les versions suivantes. PostgreSQL 13 au minimum est conseillé pour ne pas être gêné par une des limites levées dans les versions précédentes (et non développées ici).

Le partitionnement par héritage, au fonctionnement totalement différent, reste utilisable, mais ne doit plus servir aux nouveaux développements, du moins pour les cas décrits ici.

1.1 PRINCIPE & INTÉRÊTS DU PARTITIONNEMENT

- Faciliter la maintenance de gros volumes
  - `VACUUM` (`FULL`), réindexation, déplacements, sauvegarde logique…
- Performances
  - parcours complet sur de plus petites tables
  - statistiques par partition plus précises
  - purge par partitions entières
  - `pg_dump` parallélisable
  - tablespaces différents (données froides/chaudes)
- Attention à la maintenance sur le code

Maintenir de très grosses tables peut devenir fastidieux, voire impossible : `VACUUM FULL` trop long, espace disque insuffisant, autovacuum pas assez réactif, réindexation interminable… Il est aussi aberrant de conserver beaucoup de données d'archives dans des tables lourdement sollicitées pour les données récentes.

Le partitionnement consiste à séparer une même table en plusieurs sous-tables (partitions) manipulables en tant que tables à part entière.

Maintenance

La maintenance s'effectue sur les partitions plutôt que sur l'ensemble complet des données. En particulier, un `VACUUM FULL` ou une réindexation peuvent s'effectuer partition par partition, ce qui permet de limiter les interruptions en production, et lisser la charge. `pg_dump` ne sait pas paralléliser la sauvegarde d'une table volumineuse et non partitionnée, mais parallélise celle de différentes partitions d'une même table.

C'est aussi un moyen de déplacer une partie des données dans un autre *tablespace* pour des raisons de place, ou pour déporter les parties les moins utilisées de la table vers des disques plus lents et moins chers.

Parcours complet de partitions

Certaines requêtes (notamment décisionnelles) ramènent tant de lignes, ou ont des critères si complexes, qu'un parcours complet de la table est souvent privilégié par l'optimiseur.

Un partitionnement, souvent par date, permet de ne parcourir qu'une ou quelques partitions au lieu de l'ensemble des données. C'est le rôle de l'optimiseur de choisir la partition (*partition pruning*), par exemple celle de l'année en cours, ou des mois sélectionnés.

Suppression des partitions

La suppression de données parmi un gros volume peut poser des problèmes d'accès concurrents ou de performance, par exemple dans le cas de purges.

En configurant judicieusement les partitions, on peut résoudre cette problématique en supprimant une partition (`DROP TABLE nompartition ;`), ou en la *détachant* (`ALTER TABLE table_partitionnee DETACH PART` pour l'archiver (et la réattacher au besoin) ou la supprimer ultérieurement.

D'autres optimisations sont décrites dans ce billet de blog d'Adrien Nayrat[1] : statistiques plus précises au niveau d'une partition, réduction plus simple de la fragmentation des index, jointure par rapprochement des partitions…

La principale difficulté d'un système de partitionnement consiste à partitionner avec un impact minimal sur la maintenance du code par rapport à une table classique.

[1] https://blog.anayrat.info/2021/09/01/cas-dusages-du-partitionnement-natif-dans-postgresql/

1.2 PARTITIONNEMENT DÉCLARATIF

- Table partitionnée
  - structure uniquement
  - index/contraintes répercutés sur les partitions
- Partitions :
  - 1 partition = 1 table classique, utilisable directement
  - clé de partitionnement (inclue dans PK/UK)
  - partition par défaut
  - sous-partitions possibles
  - FDW comme partitions possible
  - attacher/détacher une partition

En partitionnement déclaratif, une table partitionnée ne contient pas de données par elle-même. Elle définit la structure (champs, types) et les contraintes et index, qui sont répercutées sur ses partitions.

Une partition est une table à part entière, rattachée à une table partitionnée. Sa structure suit automatiquement celle de la table partitionnée et ses modifications. Cependant, des index ou contraintes supplémentaires propres à cette partition peuvent être ajoutées de la même manière que pour une table classique.

La partition se définit par une « clé de partitionnement », sur une ou plusieurs colonnes. Les lignes de même clé se retrouvent dans la même partition. La clé peut se définir comme :

- une liste de valeurs ;
- une plage de valeurs ;
- une valeur de hachage.

Les clés des différentes partitions ne doivent pas se recouvrir.

Une partition peut elle-même être partitionnée, sur une autre clé, ou la même.

Une table classique peut être attachée à une table partitionnée. Une partition peut être détachée et redevenir une table indépendante normale.

Même une table distante (utilisant *foreign data wrapper*) peut être définie comme partition, avec des restrictions. Pour les performances, préférer alors PostgreSQL 14 au moins.

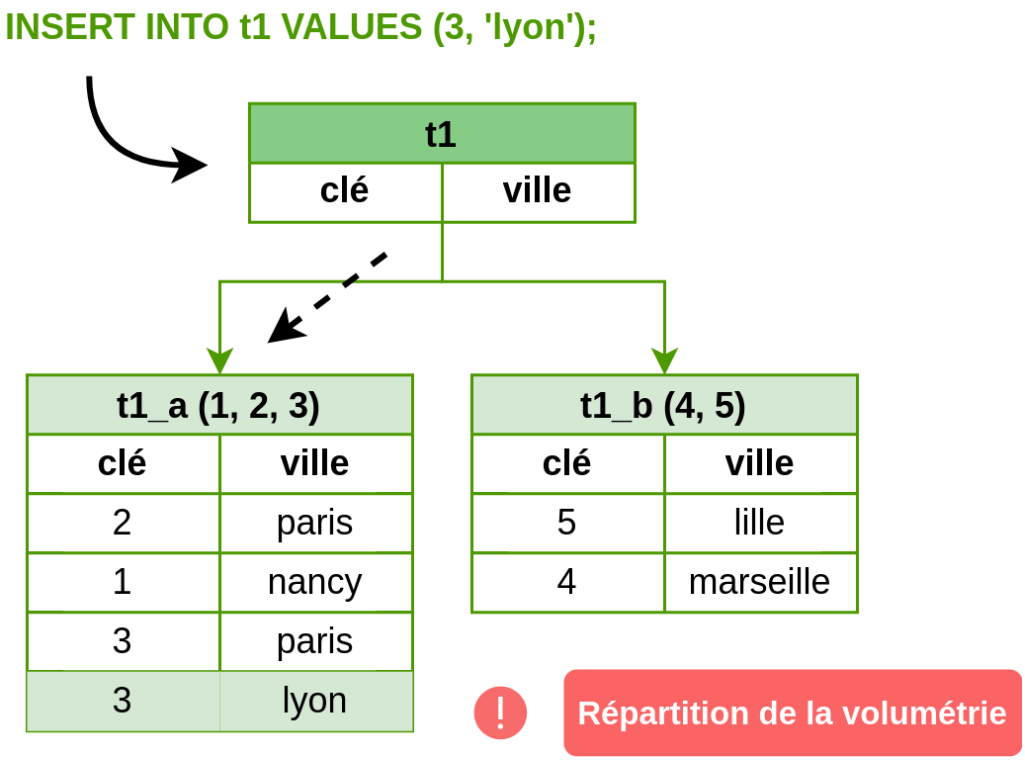
Les clés étrangères entre tables partitionnées ne sont pas un problème dans les versions récentes de PostgreSQL.

Le routage des données insérées ou modifiées vers la bonne partition est géré de façon automatique en fonction de la définition des partitions. La création d'une partition par défaut permet d'éviter des erreurs si aucune partition ne convient.

De même, à la lecture de la table partitionnée, les différentes partitions nécessaires sont accédées de manière transparente.

Pour le développeur, la table principale peut donc être utilisée comme une table classique. Il vaut mieux cependant qu'il connaisse le mode de partitionnement, pour utiliser la clé autant que possible. La complexité supplémentaire améliorera les performances. L'accès direct aux partitions par leur nom de table reste possible, et peut parfois améliorer les performances. Un développeur pourra aussi purger des données plus rapidement, en effectuant un simple `DROP` de la partition concernée.

1.2.1 Partitionnement par liste



1.2.2 Partitionnement par liste : implémentation

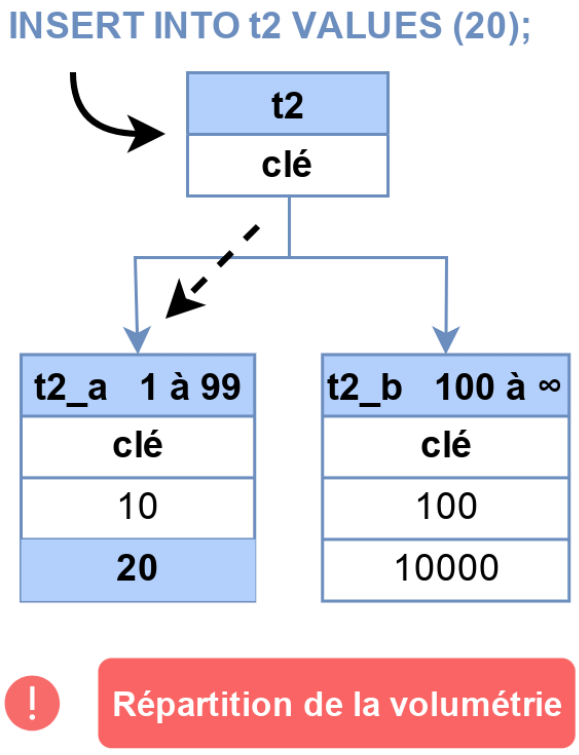
```
CREATE TABLE t1(c1 integer, c2 text) PARTITION BY LIST (c1) ;

CREATE TABLE t1_a PARTITION OF t1 FOR VALUES IN (1, 2, 3) ;
CREATE TABLE t1_b PARTITION OF t1 FOR VALUES IN (4, 5) ;
…
```

Le partitionnement par liste définit les valeurs d'une colonne acceptables dans chaque partition.

Utilisations courantes : partitionnement par année, par statut, par code géographique…

1.2.3 Partitionnement par intervalle



1.2.4 Partitionnement par intervalle : implémentation

```
CREATE TABLE logs ( d timestamptz, contenu text) PARTITION BY RANGE (d) ;

CREATE TABLE logs_201901 PARTITION OF logs
                         FOR VALUES FROM ('2019-01-01') TO ('2019-02-01');
CREATE TABLE logs_201902 PARTITION OF logs
                         FOR VALUES FROM ('2019-02-01') TO ('2019-03-01');
…
CREATE TABLE logs_201912 PARTITION OF logs
                         FOR VALUES FROM ('2019-12-01') TO ('2020-01-01');
…
CREATE TABLE logs_autres PARTITION OF logs
                         DEFAULT ;              -- pour ne rien perdre
```

Utilisations courantes : partitionnement par date, par plages de valeurs continues, alphabétiques…

L'exemple ci-dessus utilise le partitionnement par mois. Chaque partition est définie par des plages de date. Noter que la borne supérieure ne fait *pas* partie des données de la partition. Elle doit donc être aussi la borne inférieure de la partie suivante.

La description de la table partitionnée devient :

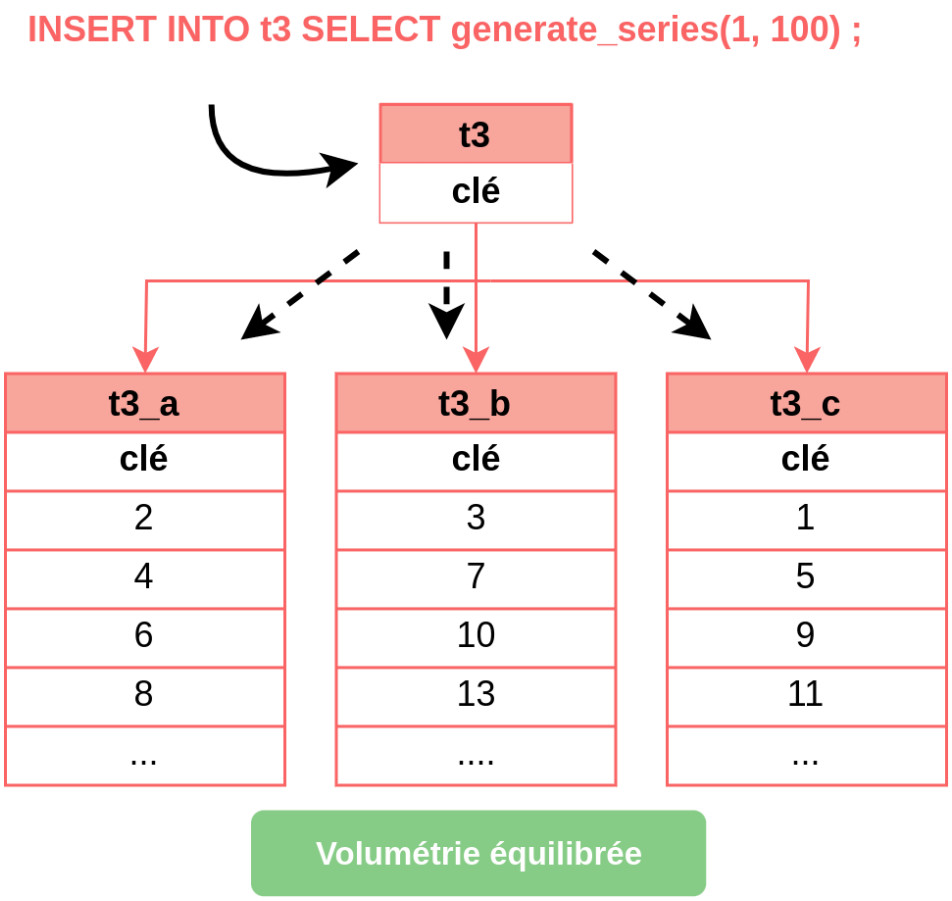
```
=# \d+ logs
                                  Table partitionnée « public.logs »
 Colonne |           Type           | ... | Stockage | …
---------+--------------------------+-----+----------+ …
 d       | timestamp with time zone | ... | plain    | …
 contenu | text                     | ... | extended | …
Clé de partition : RANGE (d)
Partitions: logs_201901 FOR VALUES FROM ('2019-01-01 00:00:00+01') TO ('2019-02-01
 ↪  00:00:00+01'),
            logs_201902 FOR VALUES FROM ('2019-02-01 00:00:00+01') TO ('2019-03-01
 ↪  00:00:00+01'),
            …
            logs_201912 FOR VALUES FROM ('2019-12-01 00:00:00+01') TO ('2020-01-01
 ↪  00:00:00+01'),
            logs_autres DEFAULT
```

La partition par défaut reçoit toutes les données qui ne vont dans aucune autre partition : cela évite des erreurs d'insertion. Il vaut mieux que la partition par défaut reste très petite.

Il est possible de définir des plages sur plusieurs champs :

```
CREATE TABLE tt_a PARTITION OF tt
FOR VALUES FROM (1,'2020-08-10') TO (100, '2020-08-11') ;
```

1.2.5 Partitionnement par hachage



1.2.6 Partitionnement par hachage : implémentation

- Hachage des valeurs
- Répartition homogène
- Indiquer un modulo et un reste

```
CREATE TABLE t3(c1 integer, c2 text) PARTITION BY HASH (c1);

CREATE TABLE t3_a PARTITION OF t3 FOR VALUES WITH (modulus 3,remainder 0);
CREATE TABLE t3_b PARTITION OF t3 FOR VALUES WITH (modulus 3,remainder 1);
CREATE TABLE t3_c PARTITION OF t3 FOR VALUES WITH (modulus 3,remainder 2);
```

Ce type de partitionnement vise à répartir la volumétrie dans plusieurs partitions de manière homogène, quand il n'y a pas de clé évidente. En général, il y aura plus que 3 partitions.

1.3 PERFORMANCES & PARTITIONNEMENT

- Insertions via la table principale
  - quasi aucun impact
- Lecture depuis la table principale
  - attention à la clé
- Purge
  - simple `DROP` ou `DETACH`
- Trop de partitions
  - attention au temps de planification

Des `INSERT` dans la table partitionnée seront redirigés directement dans les bonnes partitions avec un impact en performances quasi négligeable.

Lors des lectures ou jointures, il est important de préciser autant que possible la clé de jointure, si elle est pertinente. Dans le cas contraire, toutes les tables de la partition seront interrogées.

Dans cet exemple, la table comprend 10 partitions :

```
=# EXPLAIN (COSTS OFF)  SELECT COUNT(*) FROM pgbench_accounts ;
                                  QUERY PLAN
------------------------------------------------------------------------------
Finalize Aggregate
-> Gather
   Workers Planned: 2
   ->  Parallel Append
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_1_pkey on
 ↪  pgbench_accounts_1 pgbench_accounts
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_2_pkey on
 ↪  pgbench_accounts_2 pgbench_accounts_1
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_3_pkey on
 ↪  pgbench_accounts_3 pgbench_accounts_2
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_4_pkey on
 ↪  pgbench_accounts_4 pgbench_accounts_3
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_5_pkey on
 ↪  pgbench_accounts_5 pgbench_accounts_4
     ->  Partial Aggregate
```

```
      ->  Parallel Index Only Scan using pgbench_accounts_6_pkey on
↪  pgbench_accounts_6 pgbench_accounts_5
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_7_pkey on
↪  pgbench_accounts_7 pgbench_accounts_6
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_8_pkey on
↪  pgbench_accounts_8 pgbench_accounts_7
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_9_pkey on
↪  pgbench_accounts_9 pgbench_accounts_8
     ->  Partial Aggregate
       ->  Parallel Index Only Scan using pgbench_accounts_10_pkey on
↪  pgbench_accounts_10 pgbench_accounts_9
```

Avec la clé, PostgreSQL se restreint à la (ou les) bonne(s) partition(s) :

```
=# EXPLAIN (COSTS OFF)  SELECT * FROM pgbench_accounts WHERE aid = 599999 ;

                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using pgbench_accounts_1_pkey on pgbench_accounts_1 pgbench_accounts
   Index Cond: (aid = 599999)
```

Si l'on connaît la clé et que le développeur sait en déduire la table, il est aussi possible d'accéder directement à la partition :

```
=# EXPLAIN (COSTS OFF) SELECT * FROM pgbench_accounts_6 WHERE aid = 599999 ;

QUERY PLAN
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using pgbench_accounts_6_pkey on pgbench_accounts_6
  Index Cond: (aid = 599999)
```

Cela allège la planification, surtout s'il y a beaucoup de partitions.

Exemples :

- dans une table partitionnée par statut de commande, beaucoup de requêtes ne s'occupent que d'un statut particulier, et peuvent donc n'appeler que la partition concernée (attention si le statut change…) ;
- dans une table partitionnée par mois de création d'une facture, la date de la facture permet de s'adresser directement à la bonne partition.

Il est courant que les ORM ne sachent pas exploiter cette fonctionnalité.

1.3.1 Attacher/détacher une partition

```
ALTER TABLE logs ATTACH PARTITION logs_archives
FOR VALUES FROM (MINVALUE) TO ('2019-01-01') ;
```

- Vérification du respect de la contrainte
    - parcours complet de la table: lent + verrou !

```
ALTER TABLE logs DETACH PARTITION logs_archives ;
```

- Rapide… mais verrou

Attacher une table existante à une table partitionnée implique de définir une clé de partitionnement. PostgreSQL vérifiera que les valeurs présentes correspondent bien à cette clé. Cela peut être long, surtout que le verrou nécessaire sur la table est gênant. Pour accélérer les choses, il est conseillé d'ajouter au préalable une contrainte `CHECK` correspondant à la clé, voire d'ajouter d'avance les index qui seraient ajoutés lors du rattachement.

Détacher une partition est beaucoup plus rapide qu'en attacher une. Cependant, là encore, le verrou peut être gênant.

1.3.2 Supprimer une partition

```
DROP TABLE logs_2018 ;
```

Là aussi, l'opération est simple et rapide, mais demande un verrou exclusif.

1.3.3 Limitations principales du partitionnement déclaratif

- Temps de planification ! Max ~ 100 partitions
- Création non automatique
- Pas d'héritage multiple, schéma fixe
- Partitions distantes sans propagation d'index
- Limitations avant PostgreSQL 13/14

Certaines limitations du partitionnement sont liées à son principe. Les partitions ont forcément le même schéma de données que leur partition mère. Il n'y a pas de notion d'héritage multiple.

La création des partitions n'est pas automatique (par exemple dans un partitionnement par date). Il faudra prévoir de les créer par avance.

Une limitation sérieuse du partitionnement tient au temps de planification qui augmente très vite avec le nombre de partitions, même petites. En général, on considère qu'il ne faut pas dépasser 100 partitions.

Pour contourner cette limite, il reste possible de manipuler directement les partitions s'il est facile de trouver leur nom.

Avant PostgreSQL 13, de nombreuses limitations rendent l'utilisation moins pratique ou moins performante. Si le partitionnement vous intéresse, il est conseillé d'utiliser une version la plus récente possible.

1.4 CONCLUSION

- Préférer une version récente de PostgreSQL
- Pour plus de détails sur le partitionnement
  - https://dali.bo/v1_html

Le partitionnement déclaratif apparu en version 10 est mûr dans les dernières versions. Il introduit une complexité supplémentaire, mais peut rendre de grands services quand la volumétrie augmente.

1.5 QUIZ

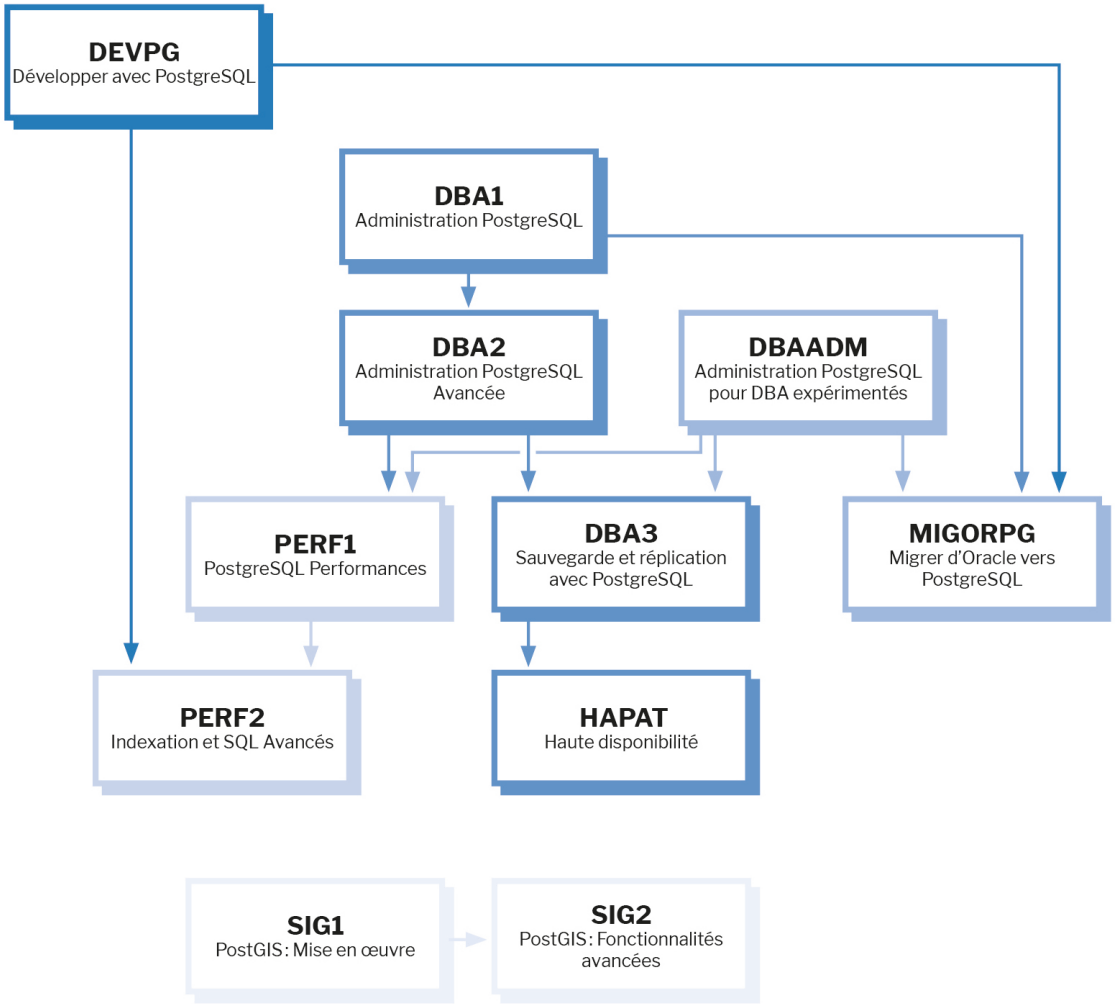
https://dali.bo/v0_quiz

Les formations Dalibo

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

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

Cursus des formations



Retrouvez nos formations dans leur dernière version :

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

Les livres blancs

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

Téléchargement gratuit

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