Module X3

Extensions PostgreSQL pour les DBA

24.04

Table des matières

Sur ce document

Formation	Module X3
Titre	Extensions PostgreSQL pour les DBA
Révision	24.04
PDF	https://dali.bo/x3_pdf
EPUB	https://dali.bo/x3_epub
HTML	https://dali.bo/x3_html
Slides	https://dali.bo/x3_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/ Extensions PostgreSQL pour les DBA

1.1 PRÉAMBULE

- Nombreuses extensions pour observer le comportement de PostgreSQL
- Contribs ou projets externes

De nombreux permettent de manipuler une facette de PostgreSQL à laquelle on n'a normalement pas accès. Leur utilisation est parfois très spécialisée et pointue.

1.2 PGSTATTUPLE

pgstattuple fournit une mesure (par parcours complet de l'objet) sur:

- Pour une table
  - remplissage des blocs
  - enregistrements morts
  - espace libre
- Pour un index
  - profondeur de l'index
  - remplissage des feuilles
  - fragmentation (feuilles non consécutives)

Par exemple :

```
# CREATE EXTENSION
# SELECT * FROM pgstattuple('dspam_token_data');

-[ RECORD 1]----------------------
 table_len          | 601743360
 tuple_count        | 8587417
 tuple_len          | 412196016
 tuple_percent      | 68.5
 dead_tuple_count   | 401098
 dead_tuple_len     | 19252704
 dead_tuple_percent | 3.2
 free_space         | 93370000
 free_percent       | 15.52

# SELECT * FROM pgstatindex('dspam_token_data_uid_key');

-[ RECORD 1]----------------------
 version            | 2
 tree_level         | 2
 index_size         | 429047808
 root_block_no      | 243
 internal_pages     | 244
 leaf_pages         | 52129
 empty_pages        | 0
 deleted_pages      | 0
 avg_leaf_density   | 51.78
 leaf_fragmentation | 43.87
```

Comme chaque interrogation nécessite une lecture complète de l'objet, ces fonctions ne sont pas à appeler en supervision.

Elles servent de façon ponctuelle pour s'assurer qu'un objet nécessite une réorganisation. Ici, l'index

`dspam_token_data_uid_key` pourrait certainement être reconstruit… il deviendrait 40 % plus petit environ (remplissage à 51 % au lieu de 90 %).

`leaf_fragmentation` indique le pourcentage de pages feuilles qui ne sont pas physiquement contiguës sur le disque. Cela peut être important dans le cas d'un index utilisé pour des `Range Scans` (requête avec des inégalités), mais n'a aucune importance ici puisqu'il s'agit d'une clé primaire technique, donc d'un index qui n'est interrogé que pour récupérer des enregistrements de façon unitaire.

1.3 PG_FREESPACEMAP

La *freespacemap* :

- est renseignée par VACUUM, par objet (table/index)
- est consommée par les sessions modifiant des données (INSERT/UPDATE)
- est interrogée la freespacemap pour connaître l'espace libre
- est rarement utilisée (doute sur l'efficacité de VACUUM)

Voici deux exemples d'utilisation de `pg_freespacemap` :

```
dspam=# SELECT * FROM pg_freespace('dspam_token_data') LIMIT 20;
 blkno | avail
-------+-------
     0 |    32
     1 |     0
     2 |     0
     3 |    32
     4 |     0
     5 |     0
     6 |     0
     7 |     0
     8 |    32
     9 |    32
    10 |    32
    11 |     0
    12 |     0
    13 |     0
    14 |     0
    15 |     0
    16 |     0
    17 |     0
    18 |    32
    19 |    32
```

```
dspam=# SELECT * FROM pg_freespace('dspam_token_data') ORDER BY avail DESC
LIMIT 20;
 blkno | avail
-------+-------
 67508 |  7520
 67513 |  7520
 67460 |  7520
 67507 |  7520
 67451 |  7520
 67512 |  7520
 67452 |  7520
 67454 |  7520
 67505 |  7520
 67447 |  7520
```

```
67324 |  7520
67443 |  7520
67303 |  7520
67509 |  7520
67444 |  7520
67448 |  7520
67445 |  7520
66888 |  7520
67516 |  7520
67514 |  7520
```

L'interprétation de « avail » est un peu complexe, et différente suivant qu'on inspecte une table ou un index. Il est préférable de se référer à la documentation.

1.4 PG_VISIBILITY

La *Visibility Map* :

- Est renseignée par `VACUUM`, par table
- Permet de savoir que l'ensemble des enregistrements de ce bloc est visible
- Indispensable pour les parcours d'index seul
- Interroger la *visibility map* permet de voir si un bloc est :
  - visible
  - gelé
- Rarement utilisé

On crée une table de test avec 451 lignes :

```
CREATE TABLE test_visibility AS SELECT generate_series(0,450) x;
SELECT 451
```

On regarde dans quel état est la *visibility map* :

```
SELECT oid FROM pg_class WHERE relname='test_visibility' ;
```

```
  oid
-------
 18370
```

```
SELECT * FROM pg_visibility(18370);
```

```
 blkno | all_visible | all_frozen | pd_all_visible
-------+-------------+------------+----------------
     0 | f           | f          | f
     1 | f           | f          | f
```

Les deux blocs que composent la table `test_visibility` sont à `false`, ce qui est normal puisque l'opération de vacuum n'a jamais été exécutée sur cette table.

On lance donc une opération de vacuum :

```
VACUUM VERBOSE test_visibility ;

INFO:  exécution du VACUUM sur « public.test_visibility »
INFO:  « test_visibility » : 0 versions de ligne supprimables,
                             451 non supprimables
parmi 2 pages sur 2
DÉTAIL : 0 versions de lignes mortes ne peuvent pas encore être supprimées.
Il y avait 0 pointeurs d éléments inutilisés.
Ignore 0 page à cause des verrous de blocs.
0 page est entièrement vide.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
```

Vacuum voit bien nos 451 lignes, et met donc la *visibility map* à jour. Lorsqu'on la consulte, on voit bien que toutes les lignes sont visibles de toutes les transactions :

```
SELECT * FROM pg_visibility(33259);
```

```
 blkno | all_visible | all_frozen | pd_all_visible
-------+-------------+------------+----------------
     0 | t           | f          | t
     1 | t           | f          | t
```

La colonne `all_frozen` passera à `t` après un `VACUUM FREEZE`.

1.5 PAGEINSPECT

- Vision du contenu d'un bloc
- Sans le dictionnaire, donc sans décodage des données
- Affichage brut
- Utilisé surtout en debug, ou dans les cas de corruption
- Fonctions de décodage pour les tables, les index (B-tree, hash, GIN, GiST), FSM
- Nécessite de connaître le code de PostgreSQL

Voici quelques exemples :

Contenu d'une page d'une table :

```
# SELECT * FROM heap_page_items(get_raw_page('dspam_token_data',0)) LIMIT 5;

 lp | lp_off | lp_flags | lp_len |   t_xmin   | t_xmax | t_field3 | t_ctid
----+--------+----------+--------+------------+--------+----------+--------
  1 |    201 |        2 |      0 |            |        |          |
  2 |   1424 |        1 |     48 | 1439252980 |      0 |        0 | (0,2)
  3 |    116 |        2 |      0 |            |        |          |
  4 |   7376 |        1 |     48 |          2 |      0 |      140 | (0,4)
  5 |   3536 |        1 |     48 | 1392499801 |      0 |        0 | (0,5)

 lp | t_infomask2 | t_infomask | t_hoff | t_bits | t_oid
----+-------------+------------+--------+--------+-------
  1 |             |            |        |        |
  2 |           5 |       2304 |     24 |        |
  3 |             |            |        |        |
  4 |           5 |      10496 |     24 |        |
  5 |           5 |       2304 |     24 |        |
```

Et son entête :

```
# SELECT * FROM page_header(get_raw_page('dspam_token_data',0));

-[ RECORD 1 ]-------------------------------------------
lsn       | F1A/5A6EAC40
checksum  | 0
flags     | 1
lower     | 852
upper     | 896
special   | 8192
pagesize  | 8192
version   | 4
prune_xid | 1450780148
```

Méta-données d'un index (contenu dans la première page) :

```
# SELECT * FROM bt_metap('dspam_token_data_uid_key');
```

```
 magic  | version | root | level | fastroot | fastlevel
--------+---------+------+-------+----------+-----------
 340322 |       2 |  243 |     2 |      243 |         2
```

La page racine est la 243. Allons la voir :

```
# SELECT * FROM bt_page_items('dspam_token_data_uid_key',243) LIMIT 10;

 offset |   ctid    | len | nulls | vars |                 data
--------+-----------+-----+-------+------+--------------------------------------
      1 | (3,1)     |   8 | f     | f    |
      2 | (44565,1) |  20 | f     | f    | f3 4b 2e 8c 39 a3 cb 80 0f 00 00 00
      3 | (242,1)   |  20 | f     | f    | 77 c6 0d 6f a6 92 db 81 28 00 00 00
      4 | (43569,1) |  20 | f     | f    | 47 a6 aa be 29 e3 13 83 18 00 00 00
      5 | (481,1)   |  20 | f     | f    | 30 17 dd 8e d9 72 7d 84 0a 00 00 00
      6 | (43077,1) |  20 | f     | f    | 5c 3c 7b c5 5b 7a 4e 85 0a 00 00 00
      7 | (719,1)   |  20 | f     | f    | 0d 91 d5 78 a9 72 88 86 26 00 00 00
      8 | (41209,1) |  20 | f     | f    | a7 8a da 17 95 17 cd 87 0a 00 00 00
      9 | (957,1)   |  20 | f     | f    | 78 e9 64 e9 64 a9 52 89 26 00 00 00
     10 | (40849,1) |  20 | f     | f    | 53 11 e9 64 e9 1b c3 8a 26 00 00 00
```

La première entrée de la page 243, correspondant à la donnée `f3 4b 2e 8c 39 a3 cb 80 0f 00 00 00` est stockée dans la page 3 de notre index :

```
# SELECT * FROM bt_page_stats('dspam_token_data_uid_key',3);

-[ RECORD 1]-----------------
blkno          | 3
type           | i
live_items     | 202
dead_items     | 0
avg_item_size  | 19
page_size      | 8192
free_size      | 3312
btpo_prev      | 0
btpo_next      | 44565
btpo           | 1
btpo_flags     | 0

# SELECT * FROM bt_page_items('dspam_token_data_uid_key',3) LIMIT 10;

 offset |   ctid    | len | nulls | vars |                 data
--------+-----------+-----+-------+------+--------------------------------------
      1 | (38065,1) |  20 | f     | f    |  f3 4b 2e 8c 39 a3 cb 80 0f 00 00 00
      2 | (1,1)     |   8 | f     | f    |
      3 | (37361,1) |  20 | f     | f    | 30 fd 30 b8 70 c9 01 80 26 00 00 00
      4 | (2,1)     |  20 | f     | f    | 18 2c 37 36 27 03 03 80 27 00 00 00
      5 | (4,1)     |  20 | f     | f    | 36 61 f3 b6 c5 1b 03 80 0f 00 00 00
      6 | (43997,1) |  20 | f     | f    | 30 4a 32 58 c8 44 03 80 27 00 00 00
      7 | (5,1)     |  20 | f     | f    | 88 fe 97 6f 7e 5a 03 80 27 00 00 00
      8 | (51136,1) |  20 | f     | f    | 74 a8 5a 9b 15 5d 03 80 28 00 00 00
      9 | (6,1)     |  20 | f     | f    | 44 41 3c ee c8 fe 03 80 0a 00 00 00
     10 | (45317,1) |  20 | f     | f    | d4 b0 7c fd 5d 8d 05 80 26 00 00 00
```

Le type de la page est `i`, c'est-à-dire « internal », donc une page interne de l'arbre. Continuons notre descente, allons voir la page 38065 :

```
# SELECT * FROM bt_page_stats('dspam_token_data_uid_key',38065);

-[ RECORD 1]----------------------
blkno          | 38065
type           | l
live_items     | 169
dead_items     | 21
avg_item_size  | 20
page_size      | 8192
free_size      | 3588
btpo_prev      | 118
btpo_next      | 119
btpo           | 0
btpo_flags     | 65

# SELECT * FROM bt_page_items('dspam_token_data_uid_key',38065) LIMIT 10;

 offset |    ctid     | len | nulls | vars |                 data
--------+-------------+-----+-------+------+--------------------------------------
      1 | (11128,118) |  20 | f     | f    | 33 37 89 95 b9 23 cc 80 0a 00 00 00
      2 | (45713,181) |  20 | f     | f    | f3 4b 2e 8c 39 a3 cb 80 0f 00 00 00
      3 | (45424,97)  |  20 | f     | f    | f3 4b 2e 8c 39 a3 cb 80 26 00 00 00
      4 | (45255,28)  |  20 | f     | f    | f3 4b 2e 8c 39 a3 cb 80 27 00 00 00
      5 | (15672,172) |  20 | f     | f    | f3 4b 2e 8c 39 a3 cb 80 28 00 00 00
      6 | (5456,118)  |  20 | f     | f    | f3 bf 29 a2 39 a3 cb 80 0f 00 00 00
      7 | (8356,206)  |  20 | f     | f    | f3 bf 29 a2 39 a3 cb 80 28 00 00 00
      8 | (33895,272) |  20 | f     | f    | f3 4b 8e 37 99 a3 cb 80 0a 00 00 00
      9 | (5176,108)  |  20 | f     | f    | f3 4b 8e 37 99 a3 cb 80 0f 00 00 00
     10 | (5466,41)   |  20 | f     | f    | f3 4b 8e 37 99 a3 cb 80 26 00 00 00
```

Nous avons trouvé une feuille (type `l`). Les ctid pointés sont maintenant les adresses dans la table :

```
# SELECT * FROM dspam_token_data WHERE ctid = '(11128,118)';

 uid |        token         | spam_hits | innocent_hits |  last_hit
-----+----------------------+-----------+---------------+------------
  40 | -6317261189288392210 |         0 |             3 | 2014-11-10
```

1.6 PGROWLOCKS

Les verrous mémoire de PostgreSQL ne verrouillent pas les enregistrements :

- Il est parfois compliqué de comprendre qui verrouille qui, à cause de quel enregistrement
- pgrowlocks inspecte une table pour détecter les enregistrements verrouillés, leur niveau de verrouillage, et qui les verrouille
- scan complet de la table !

Par exemple :

```
# SELECT * FROM pgrowlocks('dspam_token_data');
 locked_row |   locker   | multi |     xids     |       modes        | pids
------------+------------+-------+--------------+--------------------+-------
 (0,2)      | 1452109863 | f     | {1452109863} | {"No Key Update"} | {928}
```

Nous savons donc que l'enregistrement (0,2) est verrouillé par le pid 928. Nous avons le mode de verrouillage, le (ou les) numéro de transaction associés. Un enregistrement peut être verrouillé par plus d'une transaction dans le cas d'un `SELECT FOR SHARE`. Dans ce cas, PostgreSQL crée un « multixact » qui est stocké dans `locker`, `multi` vaut `true`, et `xids` contient plus d'un enregistrement. C'est un cas très rare d'utilisation.

1.7 GESTION DU CACHE

- pg_buffercache : voir ce qu'il y a dans mes *shared buffers*
- pg_prewarm : forcer le chargement du cache

pg_buffercache et pg_prewarm sont des extensions déjà connues de beaucoup de DBA.

Rappelons que pg_buffercache permet de lister chaque bloc dans le cache de PostgreSQL, et de savoir notamment s'il est *dirty*.

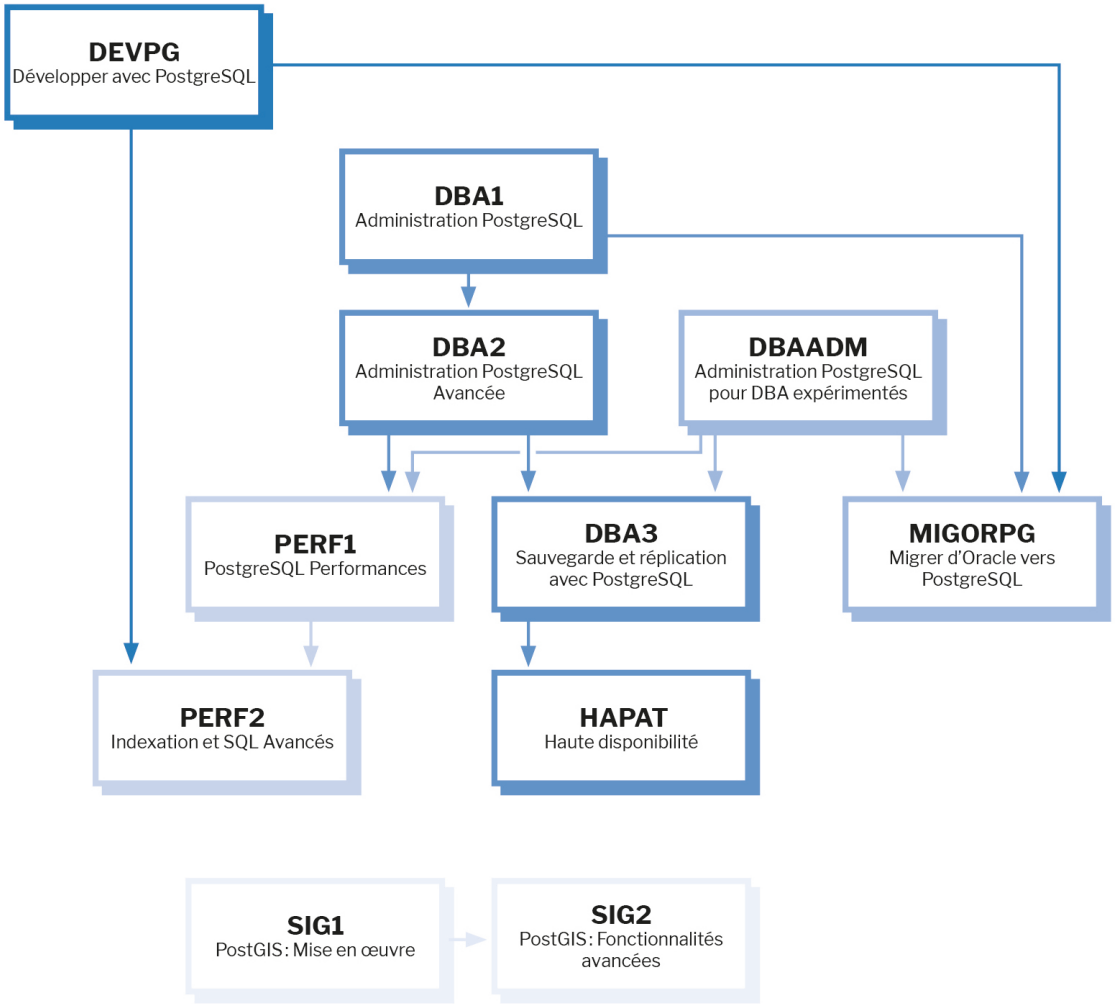
pg_prewarm est lui très utile pour forcer le chargement d'un objet dans le cache de PostgreSQL ou de l'OS, y compris automatiquement au démarrage.

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.