Workshop 12

Nouveautés de PostgreSQL 12

12

Dalibo & Contributors

https://dalibo.com/formations

Nouveautés de PostgreSQL 12

Workshop 12

TITRE : Nouveautés de PostgreSQL 12
SOUS-TITRE : Workshop 12

REVISION: 12

Table des Matières

1 NOUVEAUTÉS DE POSTGRESQL 12

Photographie de Ikiwaner[1] , licence GNU FREE Documentation Licence[2] , obtenue sur wikimedia.org[3] .

Participez à ce workshop !

Pour des précisions, compléments, liens, exemples, et autres corrections et suggestions, soumettez vos *Pull Requests* dans notre dépôt :

https://github.com/dalibo/workshops/tree/master/fr

Licence : PostgreSQL[4]

Ce workshop sera maintenu encore plusieurs mois après la sortie de la version 12.

---

[1] https://commons.wikimedia.org/wiki/User:Ikiwaner
[2] https://en.wikipedia.org/wiki/fr:Licence_de_documentation_libre_GNU
[3] https://fr.wikipedia.org/wiki/Fichier:Etosha_elefant.jpg
[4] https://github.com/dalibo/workshops/blob/master/LICENSE.md

1.1 LES NOUVEAUTÉS

- Développement SQL
- Partitionnement
- Réplication
- Monitoring
- Administration
- Performances : index
- Incompatibilités
- Fonctionnalités futures
- Ateliers

1.2 DÉVELOPPEMENT / SQL

- Colonnes générées
- Colonne `OID`
- `COMMIT` ou `ROLLBACK AND CHAIN`
- `COPY FROM WHERE`

1.2.1 AJOUT DU SUPPORT DES COLONNES GÉNÉRÉES

- Colonnes générées par une expression
- Valeur assignée à l'écriture et consignée dans la table (`STORED`)
- Plus efficace qu'un *trigger*
- Colonne en lecture seule (sauf mot clé `DEFAULT`)
- Colonnes indexables
- Mode `VIRTUAL` non supporté

Cette fonctionnalité du standard SQL, permet de créer des colonnes calculées à partir d'une expression plutôt qu'une assignation classique.

Dans le mode `STORED`, l'expression est évaluée **à l'écriture** et le résultat est consigné dans la table auprès des autres colonnes.

```
-- définition de la table

$ CREATE TABLE table1(
    id serial PRIMARY KEY,
    a int NOT NULL DEFAULT 0,
    b int NOT NULL DEFAULT 0,
    prod int generated always as (a * b) stored
) ;

$ \d table1
Table "public.table1"
Column |  Type   | Collation | Nullable |              Default
--------+---------+-----------+----------+-----------------------------------
id     | integer |           | not null | nextval('table1_id_seq'::regclass)
a      | integer |           | not null | 0
b      | integer |           | not null | 0
prod   | integer |           |          | generated always as (a * b) stored
Indexes:
"table1_pkey" PRIMARY KEY, btree (id)
```

```
-- insertion

INSERT INTO table1 (a,b) VALUES (6,7) ;
SELECT * FROM table1 ;

 id | a | b | prod
----+---+---+------
  1 | 6 | 7 |   42
(1 row)
```

Les modifications des colonnes calculées sont interdites à moins de rétablir la valeur par défaut :

```
$ UPDATE table1 SET prod = 43 ;
ERROR:  column "prod" can only be updated to DEFAULT
DETAIL:  Column "prod" is a generated column.

$ UPDATE table1 SET prod = DEFAULT ;
```

Toute mise à jour d'enregistrement étant une insertion d'un nouvel enregistrement (voir le fonctionnement du MVCC[5]), les colonnes générées sont donc recalculées quel que soit le champ modifié.

Il est tout à fait possible de créer des index utilisant des colonnes générées.

Enfin, le mode `VIRTUAL` permettant de ne pas stocker la colonne et d'évaluer l'expression **à la lecture** n'est pas encore implémenté. Ce mode est prévu dans une future version.

1.2.1.1 Restriction

- Expression `IMMUTABLE`

L'expression utilisée dans une colonne générée doit être de type *immutable*[6] , c'est à dire qu'elle doit toujours produire le même résultat pour les mêmes arguments en entrée. Certaines fonctions de PostgreSQL sont de type *volatile*[7] , comme par exemple la plupart des fonctions traitant du texte ou des dates, et lorsqu'elles dépendent de la *locale*. Il faut donc créer des fonctions intermédiaires déclarées *immutable*, faire en sorte qu'elles ne soient pas impactées par la *locale* et les utiliser en lieu et place.

---

[5] https://docs.postgresql.fr/12/mvcc.html
[6] inaltérable
[7] instable

1.2.2 VISIBILITÉ DE LA COLONNE OID

- La fin des `OID`
  - anomalies de restauration possible

Le comportement spécial de la colonne cachée `oid` a été supprimé. Cette colonne, si elle existe, est désormais visible comme toutes les autres colonnes et il n'est plus possible de créer des tables ayant ce champ spécial.

La restauration de ce type de table peut poser problème. Par exemple, **en version 11** :

```
=$ CREATE TABLE table2 (nom text, f_group int, sous_group int) WITH OIDS;
=$ INSERT INTO table2 SELECT i,i,i FROM generate_series(1,1000) i;
=$ SELECT * FROM TABLE2 LIMIT 1;

 nom  | f_group | sous_group
------+---------+------------
 1    |       1 |          1

=$ SELECT oid,* FROM TABLE2 LIMIT 1;

  oid  | nom | f_group | sous_group
-------+-----+---------+------------
 29256 | 1   |       1 |          1
```

Voici la restauration de cette base dans une **instance en v12** :

```
/usr/lib/postgresql/12/bin/pg_dump -p 5433 -t table2 postgres | psql -U postgres

pg_dump: warning: WITH OIDS is not supported anymore (table "table2")
[...]
CREATE TABLE
ALTER TABLE
COPY 1000
```

La table résultante dans la version 12 n'a plus la colonne `oid` :

```
=$ SELECT  attname FROM pg_catalog.pg_attribute
WHERE attrelid = 'public.table2'::regclass;

  attname
------------
 cmax
 cmin
 ctid
 f_group
 nom
 sous_group
```

```
 tableoid
 xmax
 xmin
(9 rows)
```

Il est important de bien utiliser la version 12 de pg_dump comme pour toute mise à jour majeure. Effectivement, l'outil gère ce cas de figure en version 12, mais naturellement pas dans les versions précédentes. Si nous restaurons sur une instance en version 12 une sauvegarde créée par le pg_dump de la version 11, nous aurions alors les erreurs suivantes :

```
/usr/lib/postgresql/11/bin/pg_dump --oids -p 5433 -t table2 postgres |
psql -U postgres
...
ERROR:  syntax error at or near "WITH"
LINE 1: COPY public.table2 (nom, f_group, sous_group) WITH OIDS FROM...
                                                      ^

invalid command \.
ERROR:  syntax error at or near "29256"
LINE 1: 29256 1 1 1
```

Notez l'utilisation de l'argument `--oids` pour inclure les valeurs des OIDs. Ici, la version 12 de PostgreSQL rejette l'option `WITH OIDS` de l'ordre `COPY FROM` et le reste de la restauration échoue.

1.2.3 COMMIT AND CHAIN

- `COMMIT AND CHAIN`
- `ROLLBACK AND CHAIN`
- Enchaîne les transactions avec les mêmes caractéristiques

Une transaction peut désormais être validée ou annulée, tout en en initiant *immédiatement* une autre, avec les mêmes caractéristiques (voir `SET TRANSACTION`[8]).

Ceci permet par exemple d'enchaîner des transactions particulières tant que cela est possible, à défaut de quoi l'ouverture de la suivante échoue.

```
$ SHOW TRANSACTION_ISOLATION
transaction_isolation
-----------------------
 read committed
```

[8] https://docs.postgresql.fr/12/sql-set-transaction.html

```
(1 row)

$ BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
$ SELECT 1;
$ COMMIT AND CHAIN ;
$ SHOW TRANSACTION_ISOLATION;
 transaction_isolation
-----------------------
 repeatable read
(1 row)

$ COMMIT;
COMMIT

$ SHOW TRANSACTION_ISOLATION;
 transaction_isolation
-----------------------
 read committed
(1 row)
```

1.2.4 COPY FROM WHERE...

- Filtrer l'import massif de données
  `COPY FROM... WHERE...`

Il est désormais possible d'ajouter une clause `WHERE` à la commande `COPY FROM` et donc de contrôler les lignes qui seront retournées.

Exemple de COPY FROM conditionnel

En reprenant la table table2 précédente :

```
$ COPY table2 TO '/tmp/table2';
$ CREATE TABLE table22 (LIKE table2 INCLUDING ALL);
$ COPY table22 FROM '/tmp/table2' WHERE f_group BETWEEN 500 AND 505;
COPY 6
```

L'insertion par `COPY` a bien sélectionné les enregistrements désirés :

```
$ TABLE table22 ;
 nom | f_group | sous_group
-----+---------+------------
 500 |     500 |        500
 501 |     501 |        501
 502 |     502 |        502
```

```
503 |      503 |      503
504 |      504 |      504
505 |      505 |      505
```

1.3 PARTITIONNEMENT

- Support des clés étrangères
- Définition du tablespace pour les partitions
- Fonctions d'information :
  - `pg_partition_root()`
  - `pg_partition_ancestors()`
  - `pg_partition_tree()`
- Nouvelle commande `\dP` pour les partitions

1.3.1 CLÉS ÉTRANGÈRES DANS LES TABLES PARTITIONNÉES

- Support des clés étrangères entre tables partitionnées

Il est désormais possible de mettre en place une clé étrangère dans une table partitionnée vers une autre table partitionnée. La relation sera établie entre la table partitionnée et toutes les partitions de la table référencée.

```
$ CREATE TABLE foo (i INT PRIMARY KEY, f FLOAT) PARTITION BY RANGE (i);
$ CREATE TABLE bar (i INT PRIMARY KEY, ifoo INT REFERENCES foo(i))
     PARTITION BY RANGE (i);

$ \d foo
              Partitioned table "public.foo"
 Column |       Type       | Collation | Nullable | Default
--------+------------------+-----------+----------+---------
 i      | integer          |           | not null |
 f      | double precision |           |          |
Partition key: RANGE (i)
Indexes:
    "foo_pkey" PRIMARY KEY, btree (i)
Referenced by:
    TABLE "bar" CONSTRAINT "bar_ifoo_fkey" FOREIGN KEY (ifoo) REFERENCES foo(i)
Number of partitions: 0

$ CREATE TABLE foo_1_5 (i INT NOT NULL, f FLOAT);
$ ALTER TABLE ONLY foo ATTACH PARTITION foo_1_5 FOR VALUES FROM (1) TO (5);

$ \d foo_1_5
                  Table "public.foo_1_5"
 Column |       Type       | Collation | Nullable | Default
--------+------------------+-----------+----------+---------
 i      | integer          |           | not null |
```

```
 f      | double precision |           |          |
Partition of: foo FOR VALUES FROM (1) TO (5)
Indexes:
    "foo_1_5_pkey" PRIMARY KEY, btree (i)
Referenced by:
    TABLE "bar" CONSTRAINT "bar_ifoo_fkey" FOREIGN KEY (ifoo) REFERENCES foo(i)
```

1.3.2 DÉFINITION DU TABLESPACE POUR LES PARTITIONS

- Gestion des tablespaces pour les tables partitionnées
  - Propagation du tablespace aux partitions filles
  - Surcharge du tablespace par partitions filles

Dans les versions précédentes, le choix des tablespace pour une table partitionnée n'était pas supporté bien que la commande `CREATE TABLE ... TABLESPACE ...` puisse être utilisée sans erreur.

À partir de la version 12, la gestion fine des tablespace est possible à n'importe quelle moment de la vie d'une table partitionnée et de ses partitions filles. Tout changement de tablespace au niveau de la table mère se propage pour les futures partitions filles ; toutes les partitions existantes doivent être déplacées une à une avec la commande `ALTER TABLE ... SET TABLESPACE`.

```
$ \! mkdir /var/lib/pgsql/tb1
$ \! mkdir /var/lib/pgsql/tb2
$ CREATE TABLESPACE tb1 LOCATION '/var/lib/pgsql/tb1/';
$ CREATE TABLESPACE tb2 LOCATION '/var/lib/pgsql/tb2/';

$ CREATE TABLE foo (i INT) PARTITION BY RANGE (i) TABLESPACE tb1;
$ CREATE TABLE foo_1_5 PARTITION OF foo FOR VALUES FROM  (1) TO (5);

$ \d foo_1_5
              Table "public.foo_1_5"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 i      | integer |           |          |
Partition of: foo FOR VALUES FROM (1) TO (5)
Tablespace: "tb1"

$ ALTER TABLE foo SET TABLESPACE tb2;
$ CREATE TABLE foo_6_10 PARTITION OF foo FOR VALUES FROM (6) TO (10) TABLESPACE tb2;

$ SELECT tablename, tablespace FROM pg_tables WHERE tablename LIKE 'foo%';
```

```
 tablename | tablespace
-----------+------------
 foo       | tb2
 foo_1_5   | tb1
 foo_6_10  | tb2
```

1.3.3 FONCTIONS D'INFORMATION SUR LE PARTITIONNEMENT

- `pg_partition_root(regclass)`
- `pg_partition_ancestors(regclass)`
- `pg_partition_tree(regclass)`

Trois nouvelles fonctions permettent de récupérer les informations d'un partitionnement à partir de la table mère ou à partir d'une des partitions.

`pg_partition_root` renvoie la partition mère d'une partition, `pg_partition_ancestors` renvoie la partition mère ainsi que la partition concernée, `pg_partition_tree` renvoie tout l'arbre de la partition sous forme de tuples

---

1.3.4 COMMANDE PSQL POUR LES TABLES PARTITIONNÉES

- Commande `\dP[tin+] [PATTERN]`

Le client psql est maintenant doté d'une commande rapide pour lister les tables partitionnées :

```
\dP
List of partitioned relations
Schema |   Name    |  Owner   |       Type        | Table
--------+-----------+----------+-------------------+-------
public | foo       | postgres | partitioned table |
public | bar       | postgres | partitioned table |
public | foo_pkey  | postgres | partitioned index | foo
public | bar_pkey  | postgres | partitioned index | bar
```

1.4 RÉPLICATION

- Nouveauté des `postgresql.conf`et `recovery.conf`
- 2 fichiers *trigger*
- Paramètres modifiables à chaud
- Fonction `pg_promote()`
- Copie de slot de réplication

1.4.1 NOUVEAUTÉ DE `POSTGRESQL.CONF` ET `RECOVERY.CONF`

- `recovery.conf` disparaît
- Tous les paramètres dans `postgresql.conf`

Avec la version 12 de PostgreSQL, le fichier `recovery.conf` disparaît. Les paramètres de l'ancien fichier `recovery.conf` sont dans le fichier `postgresql.conf`.

Si le fichier `recovery.conf` est présent, PostgreSQL refuse de démarrer.

```
FATAL:  using recovery command file "recovery.conf" is not supported
LOG:  startup process (PID 22810) exited with exit code 1
LOG:  aborting startup due to startup process failure
LOG:  database system is shut down
```

Les paramètres de l'ancien `recovery.conf` se retrouvent dans le fichier `postgresql.conf`, dans 2 sections.

Section pour les paramètres concernant le mode de recovery.

```
#------------------------------------------------------------------------------
# WRITE-AHEAD LOG
#------------------------------------------------------------------------------

# - Archive Recovery -
# These are only used in recovery mode.

restore_command = '/opt/pgsql/12b2/bin/pg_standby /opt/pgsql/archives %f %p %r'
                                # command to use to restore $
                                # placeholders: %p = path of file to restore
                                #               %f = file name only
                                # e.g. 'cp /mnt/server/archivedir/%f %p'
                                # (change requires restart)
archive_cleanup_command = '/opt/pgsql/12b2/bin/pg_archivecleanup
                               -d /opt/pgsql/archives %r'
                                # command to execute$
```

```
#recovery_end_command = ''      # command to execute at completion of recovery

# - Recovery Target -

# Set these only when performing a targeted recovery.

#recovery_target = ''           # 'immediate' to end recovery as soon as a
                                # consistent state is reached
                                # (change requires restart)
#recovery_target_name = ''      # the named restore point to which recovery will proceed
                                # (change requires restart)
#recovery_target_time = ''      # the time stamp up to which recovery will proceed
                                # (change requires restart)
#recovery_target_xid = ''       # the transaction ID up to which recovery will proceed
                                # (change requires restart)
#recovery_target_lsn = ''       # the WAL LSN up to which recovery will proceed
                                # (change requires restart)
#recovery_target_inclusive = on # Specifies whether to stop:
                                # just after the specified recovery target (on)
                                # just before the recovery target (off)
                                # (change requires restart)
#recovery_target_timeline = 'latest'    # 'current', 'latest', or timeline ID
                                        # (change requires restart)
#recovery_target_action = 'pause'       # 'pause', 'promote', 'shutdown'
                                        # (change requires restart)
```

Section pour les paramètres concernant la configuration des réplicats.

```
#------------------------------------------------------------------------------
# REPLICATION
#------------------------------------------------------------------------------

# - Standby Servers -
# These settings are ignored on a master server.

#primary_conninfo = ''                  # connection string to sending server
                                        # (change requires restart)
#primary_slot_name = ''                 # replication slot on sending server
                                        # (change requires restart)
#promote_trigger_file = ''              # file name whose presence ends recovery
#hot_standby = on                       # "off" disallows queries during recovery
                                        # (change requires restart)
#max_standby_archive_delay = 30s        # max delay before canceling queries
                                        # when reading WAL from archive;
                                        # -1 allows indefinite delay
#max_standby_streaming_delay = 30s      # max delay before canceling queries
                                        # when reading streaming WAL;
```

```
                                        # -1 allows indefinite delay
#wal_receiver_status_interval = 10s     # send replies at least this often
                                        # 0 disables
#hot_standby_feedback = off             # send info from standby to prevent
                                        # query conflicts
#wal_receiver_timeout = 60s             # time that receiver waits for
                                        # communication from master
                                        # in milliseconds; 0 disables
#wal_retrieve_retry_interval = 5s       # time to wait before retrying to
                                        # retrieve WAL after a failed attempt
#recovery_min_apply_delay = 0           # minimum delay for applying changes during recovery
```

Paramètres renommés ou supprimés :

- `standby_mode` a été supprimé des paramètres et est remplacé par un fichier trigger sur disque.
- `trigger_file` a été renommé en `promote_trigger_file`.

1.4.2 2 FICHIERS TRIGGER

- `standby.signal`
- `recovery.signal`

Pour que PostgreSQL démarre en mode **standby** ou **recovery**, 2 fichiers trigger sont utilisés, ils sont à positionner à la racine de l'instance PostgreSQL.

- `standby.signal` : (remplace le paramètre `standby_mode=on`) permet de configurer l'instance en instance de secours.
- `recovery.signal` : permet de configurer l'instance en mode récupération (exemple : restauration PITR).

1.4.3 PARAMÈTRES MODIFIABLES À CHAUD

- `archive_cleanup_command`
- `recovery_end_command`
- `recovery_min_apply_delay`
- `promote_trigger_file`

Les paramètres suivants, sont modifiables à chaud :

- `archive_cleanup_command` : permet de nettoyer les WAL qui ont été rejoués sur l'instance secondaire.
- `recovery_end_command` : permet de spécifier une commande (shell) à exécuter une fois l'instance restaurée.
- `recovery_min_apply_delay` : permet de différer l'application des WAL sur l'instance secondaire
- `promote_trigger_file` : permet de spécifier le chemin du fichier dont la présence déclenche la promotion de l'instance en standby.

1.4.4 FONCTION PG_PROMOTE()

- `pg_promote`

PostgreSQL 12 offre la possibilité de promouvoir une instance standby (hot_standby) à l'aide d'une fonction `psql`.

Rappel des 2 possibilités de promotion en version 11 :

- commande système : `pg_ctl promote`
- création du fichier "trigger" en commande shell, en ayant préalablement configuré le paramètre `trigger_file` dans le fichier `recovery.conf`.

Dans les 2 cas, il faut avoir accès au système de fichiers avec les droits `postgres`.

PostgreSQL 12 offre un troisième moyen de déclencher une promotion avec une fonction système SQL.

Énorme avantage : il n'est pas nécessaire de se connecter physiquement au serveur pour déclencher la promotion d'une standby. On notera que cela nécessite que le serveur soit en capacité d'accepter des connexions et donc accessible en lecture (hot_standby).

Par défaut, la fonction `pg_promote()` attend la fin de la promotion pour renvoyer le résultat à l'appelant, avec un timeout max de 60 secondes. La fonction `pg_promote()` accepte 2 paramètres optionnels :

- `wait` (booléen) : permet de ne pas attendre le résultat de la promotion (true par défaut)
- `wait_seconds` : permet de renvoyer le résultat après ce délai (par défaut : 60 secondes)

La fonction renvoie `true`, si la promotion s'est bien déroulée et `false` sinon.

Exemple d'utilisation :

```
-- Par défaut sans paramètres
postgres=# SELECT pg_promote();
 pg_promote
------------
 t
-- Attend au plus 10 secondes
postgres=# SELECT pg_promote(true,10);
-- Pas d'attente
postgres=# SELECT pg_promote(false);
```

L'accès à la fonction `pg_promote()` est limité aux super-utilisateurs. À noter qu'il est possible de déléguer les droits à un autre utilisateur ou un autre rôle :

```
postgres=# GRANT EXECUTE ON FUNCTION pg_promote TO mon_role;
GRANT
```

La fonction exécutée sur une instance non standby renvoie une erreur.

```
postgres=# select pg_promote(true,10);
psql: ERROR:  recovery is not in progress
HINT:  Recovery control functions can only be executed during recovery.
```

1.4.5 COPIE DE SLOT DE RÉPLICATION

- `pg_copy_physical_replication_slot('slot1','slot2')`

Rappel historique des slots de réplication :

- Introduction du slot de réplication physique en version 9.4 pour éviter la suppression des WAL sur l'instance primaire alors qu'un réplicat est arrêté ou trop en retard sur sa réplication.
- Le slot de réplication logique a été introduit en version PostgreSQL 10.
- PostgreSQL 12 permet grâce à 2 fonctions de copier les slots de réplications.

Cas d'usage de ces fonctions :

Attacher 2 réplicats à la même instance principale en utilisant 2 slots de réplication physique différents. Les réplicats sont réalisés à partir du même backup, pour gagner du temps et de la place, et commencent par le même LSN.

L'utilitaire `pg_basebackup` est utilisé pour créer la sauvegarde et les slots de réplication en même temps.

Note : l'argument `--write-recovery-conf` en version PostgreSQL 12 créera un fichier `standby.signal` et modifiera le fichier `postgresql.auto.conf`.

```
postgres@workshop12:~/12/data$ mkdir -p /opt/pgsql/backups
postgres@workshop12:~/12/data$ pg_basebackup --slot physical_slot1 \
--create-slot --write-recovery-conf -D /opt/pgsql/backups/

postgres@workshop12:~/12/data$ psql -x -c \
"SELECT * FROM pg_replication_slots"

-[ RECORD 1 ]-------+---------------
slot_name           | physical_slot1
plugin              |
slot_type           | physical
datoid              |
database            |
temporary           | f
active              | f
active_pid          |
xmin                |
catalog_xmin        |
restart_lsn         | 0/9000000
confirmed_flush_lsn |
```

Copie du slot de réplication

```
postgres@workshop12:~/12/repl_1$ psql -c \
"SELECT pg_copy_physical_replication_slot('physical_slot1','physical_slot2')"

pg_copy_physical_replication_slot
------------
 (physical_slot2,)

postgres@workshop12:~/12/repl_1$ psql -c \
"select slot_name,restart_lsn,slot_type,active from pg_replication_slots"

   slot_name    | restart_lsn | slot_type | active
----------------+-------------+-----------+--------
 physical_slot1 | 0/9000000   | physical  | f
 physical_slot2 | 0/9000000   | physical  | f
(2 rows)

rsync -r -p /opt/pgsql/backups/ /opt/pgsql/12/repl_1
```

```
rsync -r -p /opt/pgsql/backups/ /opt/pgsql/12/repl_2
```

Modification du slot de réplication sur le réplicat 2 en éditant le fichier `postgresql.auto.conf` et en modifiant le nom du slot.

```
primary_slot_name = 'physical_slot2'
```

Changer les ports des réplicats en éditant les fichiers `postgresql.conf`.

```
port=5434 # pour le premier réplicat
port=5435 # pour le second réplicat
```

Démarrage des instances répliquées.

```
postgres@workshop12:~/12$ pg_ctl -D /opt/pgsql/12/repl_1 start
```

Vérification des slots sur le primaire : on voit ici que le premier réplicat est actif avec le LSN A000148

```
postgres@workshop12:~/12$ psql -c \
"select slot_name,restart_lsn,slot_type,active from pg_replication_slots"

   slot_name    | restart_lsn | slot_type | active
----------------+-------------+-----------+--------
 physical_slot1 | 0/A000148   | physical  | t
 physical_slot2 | 0/9000000   | physical  | f
```

Vérification de l'accès au réplicat 1.

```
workshop12:~/12$ psql -p5434 -c "select pg_is_in_recovery()"
pg_is_in_recovery
------------
 t
```

Même chose pour le réplicat 2 .

Démarrage de l'instance

```
postgres@workshop12:~/12$ pg_ctl -D /opt/pgsql/12/repl_2 start
```

Vérification sur le primaire : on voit ici que le réplicat 2 est maintenant actif avec le même LSN que le réplicat 1.

```
postgres@workshop12:~/12$ psql -c \
"select slot_name,restart_lsn,slot_type,active from pg_replication_slots"

   slot_name    | restart_lsn | slot_type | active
----------------+-------------+-----------+--------
 physical_slot1 | 0/A000148   | physical  | t
```

```
 physical_slot2 | 0/A000148   | physical  | t
```

Vérification de l'accès au réplicat 2.

```
postgres@workshop12:~/12$ psql -p5435 -c \
"select pg_is_in_recovery()"

pg_is_in_recovery
------------
 t
```

1.5 MONITORING

- Échantillon des requêtes dans les logs
- Vues de progression pour `CREATE INDEX`, `CLUSTER`, `VACUUM`
- Listing :
  - des fichiers dans les répertoires `status` des archives des *wals*
  - des fichiers temporaires
- `pg_stat_replication` : timestamp du dernier message reçu du secondaire

1.5.1 ÉCHANTILLON DES REQUÊTES DANS LES LOGS

- `log_transaction_sample_rate`

Adrien Nayrat a soumis un correctif proposant l'échantillonnage des transactions dans les journaux d'activité.

`log_transaction_sample_rate`, dont la valeur doit être comprise entre 0 et 1.0, définit la fraction des transactions dont les opérations sont toutes tracées, en plus de celles tracées pour d'autres raisons. Il s'applique à chaque nouvelle transaction quelle que soit la durée de ses opérations. La valeur par défaut 0 désactive cette fonctionnalité alors que la valeur 1 enregistre tous les ordres pour toutes les transactions.

1.5.2 INFORMATIONS DE PROGRESSION

- Nouvelles vues pour l'avancement des tâches de maintenance
  - `pg_stat_progress_cluster`
  - `pg_stat_progress_create_index`
- En complément d'une déjà existante depuis la version 11
  - `pg_stat_progress_vacuum`

1.5.3 PROGRESSION DES RÉÉCRITURES DE TABLE

- Vue `pg_stat_progress_cluster`
  - Pour les opérations `CLUSTER` et `VACCUM FULL`

Lors de l'opération `CLUSTER`[9] et `VACUUM FULL`, la vue `pg_stat_progress_cluster` indique la progression de l'opération qui dans certains cas, peut être très longue.

On lance le traitement dans une session :

```
$ CLUSTER tab USING tab_i_j_idx;
CLUSTER
```

On observe la progression dans une autre session :

```
$ SELECT
  phase,
  heap_tuples_scanned,
  heap_tuples_written
  FROM
    pg_stat_progress_cluster;
$ \watch 1
```

```
          jeu. 17 oct. 2019 18:20:05 CEST (every 1s)

        phase        | heap_tuples_scanned | heap_tuples_written
---------------------+---------------------+---------------------
 index scanning heap |              890880 |              890880
(1 row)

           jeu. 17 oct. 2019 18:20:06 CEST (every 1s)

        phase        | heap_tuples_scanned | heap_tuples_written
---------------------+---------------------+---------------------
 index scanning heap |             1640280 |             1640280

...

         jeu. 17 oct. 2019 18:20:23 CEST (every 1s)

      phase       | heap_tuples_scanned | heap_tuples_written
------------------+---------------------+---------------------
 rebuilding index |            11000000 |            11000000
```

[9] https://docs.postgresql.fr/12/sql-cluster.html

```
(1 row)

...

   jeu. 17 oct. 2019 18:20:47 CEST (every 1s)

 phase | heap_tuples_scanned | heap_tuples_written
-------+---------------------+---------------------
(0 rows)
```

1.5.4 PROGRESSION DES MAINTENANCES D'INDEX

- Vue `pg_stat_progress_create_index`
  - Pour les opérations `CREATE INDEX` et `REINDEX`

Lors de la création d'index, la progression est consultable dans la vue `pg_stat_progress_create_index` :

```
$ CREATE INDEX ON a_table (i, a, b);
```

Dans une autre session, avant de lancer la création de l'index :

```
SELECT
  datname,
  relid::regclass,
  command,
  phase,
  tuples_done
FROM
  pg_stat_progress_create_index;


          ven. 26 juil. 2019 17:41:18 CEST (every 1s)

 datname | relid | index_relid | command | phase | tuples_done
---------+-------+-------------+---------+-------+-------------
(0 rows)
$ \watch 1

                            ven. 26 juil. 2019 17:41:19 CEST (every 1s)

 datname  |  relid  |   command    |                 phase                  | tuples_done
----------+---------+--------------+----------------------------------------+-------------
 postgres | a_table | CREATE INDEX | building index: loading tuples in tree |      718515
```

```
(1 row)

                                ven. 26 juil. 2019 17:41:20 CEST (every 1s)

 datname  |  relid  |   command    |                 phase                  | tuples_done
----------+---------+--------------+----------------------------------------+-------------
 postgres | a_table | CREATE INDEX | building index: loading tuples in tree |     1000000
(1 row)
```

1.5.5 ARCHIVE STATUS

- `pg_ls_archive_statusdir()`

Liste le nom, taille et l'heure de la dernière modification des fichiers dans le dossier `status` de l'archive des *WAL*. Il faut être membre du group `pg_monitor` ou avoir explicitement le droit (`pg_read_server_files`).

À savoir que ce répertoire est peuplé lorsqu'un journal de transactions (wal) est archivé par l'`archive_command`.

```
$ SELECT pg_switch_wal ();
$ SELECT pg_switch_wal ();
$ SELECT pg_switch_wal ();
$ SELECT * FROM pg_ls_archive_statusdir ();
```

```
             name               | size |      modification
--------------------------------+------+------------------------
 000000010000000100000000CB.done |    0 | 2019-09-03 11:51:39+02
 000000010000000100000000CD.done |    0 | 2019-09-09 14:14:48+02
 000000010000000100000000CC.done |    0 | 2019-09-09 14:14:48+02
(3 rows)
```

1.5.6 FICHIERS TEMPORAIRES

- `pg_ls_tmpdir()`

La supervision des fichiers temporaires est désormais possible dans une session :

```
$ select * from pg_ls_tmpdir();
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8686.4 | 1073741824 | 2019-09-09 14:18:29+02
 pgsql_tmp8686.3 | 1073741824 | 2019-09-09 14:18:19+02
 pgsql_tmp8686.2 | 1073741824 | 2019-09-09 14:18:13+02
 pgsql_tmp8686.5 |  456941568 | 2019-09-09 14:18:31+02
 pgsql_tmp8686.1 |   26000000 | 2019-09-09 14:17:56+02
 pgsql_tmp8686.0 | 1073741824 | 2019-09-09 14:18:05+02
(6 rows)
```

1.5.7 AJOUT DANS LA VUE `PG_STAT_REPLICATION`

- timestamp du dernier message reçu du standby

Dans le cadre de la supervision de la réplication, il est possible de déterminer l'heure à laquelle un secondaire en standby a communiqué pour la dernière fois, avec le primaire.

```
$ SELECT * FROM pg_stat_replication;
```

1.6 ADMINISTRATION

- Nouveautés sur le `VACUUM`
- Recyclage des *WAL*
- Environnement client : `PG_COLORS`, `EXPLAIN SETTINGS`
- Outils : `pg_upgrade`, `pg_ctl`, `pg_checksums`
- Paramètres de `postgresql.conf`

1.6.1 NOUVEAUTÉS DU VACUUM

- `VACUUM TRUNCATE`
- `SKIP_LOCKED`
- `INDEX_CLEANUP`
- Nouvelles options de `vacuumdb`

1.6.2 VACUUM TRUNCATE

- Indique si l'espace en fin de table doit être libéré
- Évite un verrou exclusif en fin de traitement
- Nouvel attribut de table : `VACUUM_TRUNCATE`
- Nouvelle option de la commande : `VACUUM (TRUNCATE on)`
- Non disponible pour `vacuumdb`

Cette nouvelle fonctionnalité de `VACUUM` permet de contrôler si l'opération doit tronquer l'espace vide en fin de la table. C'est le fonctionnement historique de `VACUUM` et il est par conséquent conservé par défaut.

Tronquer une table en fin de commande est extrêmement rapide, mais nécessite que la commande VACUUM acquiert un verrou exclusif sur la table le temps d'effectuer l'opération. Cette prise de verrou est parfois impossible à cause de l'activité sur la table ou peut être jugée trop gênante lors d'une maintenance ponctuelle effectuée par l'administrateur.

Ce comportement peut être modifié indépendamment pour chaque table grâce à l'attribut `VACUUM_TRUNCATE`. Par exemple:

```
ALTER TABLE matable SET (VACUUM_TRUNCATE=OFF);
```

L'option TRUNCATE a également été ajoutée à la commande SQL VACUUM. L'exemple suivant permet de ne pas tronquer la table à la fin du vacuum.

```
pg12=# VACUUM (TRUNCATE OFF) t1;
VACUUM
```

Il n'existe pour le moment pas d'argument équivalent pour la commande vacuumdb.

1.6.3 SKIP_LOCK

- Plus d'attente de verrou
- Concerne VACUUM et ANALYZE

Jusqu'en version 11, en cas de conflit de verrous sur une table, les commandes VACUUM ou ANALYZE attendaient que le verrou conflictuel soit levé pour débuter leur traitement.

Depuis la version 12 l'option SKIP_LOCKED permet d'ignorer les tables sur lesquelles un des verrous présents empêche l'exécution immédiate de la commande. Si une table est ignorée pour cette raison, un message d'avertissement (WARNING) est émis.

Exemple :

```
pg12=# VACUUM (SKIP_LOCKED ON) t1;
psql: WARNING:  skipping vacuum of "t1" --- lock not available
VACUUM
```

1.6.4 VACUUM INDEX CLEANUP

- Nouvel attribut permettant de ne pas nettoyer les index

Sur la commande VACUUM, l'option INDEX_CLEANUP OFF permet de désactiver le parcours des index lors du vacuum. Dans le cas où l'option n'est pas spécifiée, le processus prendra en compte le paramètre VACUUM_INDEX_CLEANUP de la table. Par défaut VACUUM_INDEX_CLEANUP est à on.

Cette option est surtout utile pour effectuer des opérations ponctuelles où la commande VACUUM doit être la plus rapide et légère possible. Par exemple, pour mettre à jour la visibility map ou effectuer un freeze de la table. Il est déconseillé de désactiver cette option durablement sur des tables, au risque de voir les performances de leurs index décroître fortement avec le temps.

Voici un exemple d'utilisation:

```
bench=# VACUUM (VERBOSE ON, INDEX_CLEANUP OFF) pgbench_accounts ;
INFO:  vacuuming "public.pgbench_accounts"
INFO:  "pgbench_accounts": found 0 removable, 497543 nonremovable row versions
in 8169 out of 16406 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 1253
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.11 s, system: 0.01 s, elapsed: 0.13 s.
VACUUM
Time: 146,705 ms


bench=# VACUUM (VERBOSE ON, INDEX_CLEANUP ON) pgbench_accounts ;
INFO:  vacuuming "public.pgbench_accounts"
INFO:  scanned index "pgbench_accounts_pkey" to remove 735 row versions
DETAIL:  CPU: user: 0.09 s, system: 0.00 s, elapsed: 0.09 s
INFO:  "pgbench_accounts": removed 735 row versions in 735 pages
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "pgbench_accounts_pkey" now contains 1000000 row versions in 2745 pages
DETAIL:  735 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
INFO:  "pgbench_accounts": found 0 removable, 497543 nonremovable row versions
in 8169 out of 16406 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 1256
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.22 s, system: 0.01 s, elapsed: 0.23 s.
VACUUM
```

Le paramètre `VACUUM_INDEX_CLEANUP` peut être configuré à `OFF` sur une table pour désactiver les parcours d'index lors des vacuum.

```
pg12=# ALTER TABLE t1 SET (VACUUM_INDEX_CLEANUP=OFF);
ALTER TABLE
pg12=# \d+ t1
                                    Table "public.t1"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 id     | integer |           |          |         | plain   |              |
Access method: heap
Options: vacuum_index_cleanup=off
```

Il n'est pas possible de désactiver ce parcours du vacuum index par index.

1.6.5 NOUVELLES OPTIONS DE VACUUMDB

- `--min-xid-age`
- `--min-mxid-age`
- `--disable-page-skipping`
- `--skip-locked`

PostgreSQL 12 offre 4 nouveaux arguments pour la commande `vacuumdb`:

- `--min-xid-age=XID_AGE`

  Permet à l'administrateur de traiter en priorité les tables dont l'age s'approche de la valeur de `autovacuum_freeze_max_age`. Cette action évite que l'opération ne soit traitée avec un `vacuum to prevent wraparound` à l'initiative de l'autovacuum.

- `--min-mxid-age=MXID_AGE`

  Permet à l'administrateur de traiter en priorité les tables dont l'age de la plus ancienne multixact s'approche de la valeur de `autovacuum_multixact_freeze_max_age`. Cette action évite que l'opération ne soit traitée avec un `vacuum to prevent wraparound` à l'initiative de l'autovacuum.

- `--disable-page-skipping`

  Permet de lancer un `VACUUM (DISABLE_PAGE_SKIPPING ON)` à partir de la ligne de commande (versions 9.6 et supérieures de PostgreSQL). Dans ce mode, on effectuera un nettoyage de tous les tuples, y compris s'ils sont freezés, visibles par toutes les transactions ou verrouillés. Ce mode est à utiliser en cas de suspicion de corruption des données.

- `--skip-locked`

  Ignore les tables verrouillées (versions PostgreSQL 12 et supérieures).

1.6.6 JOURNAUX DE TRANSACTIONS

Nouvelles options pour les journaux de transactions :

- `wal_recycle`
- `wal_init_zero`

- `wal_recycle` (défaut = `on`)

  Les fichiers *WAL* sont recyclés en renommant les anciens WAL évitant ainsi la création de nouveaux fichiers, opération souvent plus lente. Configurer le paramètre

`wal_recycle` à `off` permet d'obliger PostgreSQL à créer de nouveaux fichiers lors du recyclage des WAL et à supprimer les anciens. Ce mode est plus performant sur certains systèmes de fichiers de type Copy-On-Write (eg. ZFS ou BTRFS).

- `wal_init_zero` (défaut = `on`)

  Les nouveaux fichiers WAL sont remplis de zéros à la création. Cela garantit que l'espace est alloué pour le fichier WAL avant d'écrire dedans. Si `wal_init_zero` est à `off`, seul l'octet final est écrit afin de s'assurer que le fichier a bien la taille requise.

1.6.7 ENVIRONNEMENT CLIENT

- Variables d'environnement `PG_COLOR` et `PG_COLORS`
- Formatage *CSV* en sortie de `psql`
- `EXPLAIN (SETTINGS)`

1.6.8 VARIABLES D'ENVIRONNEMENT PG_COLOR ET PG_COLORS

Les nouvelles variables d'environnement (client) `PG_COLOR` et `PG_COLORS` permettent d'ajouter et de personnaliser une coloration des erreurs, warning et mots clés à la sortie de la commande `psql`.

- PG_COLOR permet de configurer la coloration des messages.

Les valeurs possibles pour `PG_COLOR` sont `always`, `auto`, `never`.

Exemple :

```
$ export PG_COLOR=always
```

- PG_COLORS permet de personnaliser la couleur des messages en fonction de leur catégorie.

Les valeurs par défaut des couleurs sont :

Catégorie	Valeur par défaut
error	01;31 (rouge)
warning	01;35 (mauve)
locus	01 (gras)

Pour modifier les couleurs, la syntaxe est la suivante :

```
$ export PG_COLORS='error=01;33:warning=01;31:locus=03'
```

Cet exemple permet de colorer les `error` en vert et les `warning` en bleu, et les mots clés en italique.

1.6.9 FORMAT CSV

PostgreSQL 12 permet de formater la sortie de la commande psql au format CSV.

Il existe 2 façons de changer le format de sortie :

1 - Ajouter l'argument `--csv` à la ligne de commande `psql`

Exemple :

```
$ psql --csv -c "select name,setting,source,boot_val from pg_settings limit 1"
name,setting,source,boot_val
allow_system_table_mods,off,default,off
```

2 - Exécuter `\pset format csv` dans l'interpréteur `psql`.

Exemple :

```
pg12=# select name,setting,source,boot_val from pg_settings limit 1;
          name           | setting | source  | boot_val
-------------------------+---------+---------+----------
 allow_system_table_mods | off     | default | off
(1 row)

pg12=# \pset format csv
Output format is csv.

pg12=# select name,setting,source,boot_val from pg_settings limit 1;
name,setting,source,boot_val
allow_system_table_mods,off,default,off
```

1.6.10 OPTION SETTINGS POUR EXPLAIN

L'option `SETTINGS ON` spécifiée lors de l'instruction `EXPLAIN` permet de générer des informations sur les paramètres modifiés au cours de la session ou de la transaction et qui influencent l'exécution de la requête étudiée.

```
pg12=# SET enable_sort = off;
SET
pg12=# EXPLAIN (SETTINGS ON) SELECT * FROM t1 WHERE id = 50;
                      QUERY PLAN
-------------------------------------------------
 Seq Scan on t1  (cost=0.00..2.25 rows=1 width=4)
```

```
   Filter: (id = 50)
 Settings: enable_sort = 'off'
```

1.6.11 OUTILS

- `pg_upgrade --clone` et `pg_upgrade --socketdir`
- Rotation des logs avec `pg_ctl logrotate`
- `pg_checksums`, anciennement `pg_verify_checksums`

1.6.12 NOUVEAUTÉS DE PG_UPGRADE

Deux nouvelles options ont été ajoutées à la commande `pg_upgrade`.

- `--clone`

  Ce paramètre permet à la commande `pg_upgrade` d'effectuer un clonage à l'aide des liens *reflink*. L'utilisation de ce paramètre dépend du système d'exploitation et du système de fichiers.

- `--socketdir` ou `-s`

  Ce paramètre permet de spécifier un répertoire pour la création d'une socket locale.

1.6.13 ROTATION DES TRACES AVEC PG_CTL

Jusqu'en version 11, la rotation des traces se faisait en envoyant un signal `SIGUSR1` au processus `logger` ou grâce à la fonction SQL `pg_rotate_logfile()`.

Depuis PostgreSQL 12, le mode `logrotate` a été ajouté à la commande `pg_ctl`.

Exemple :

```
postgres@workshop12:~/12/data$ pg_ctl -D /opt/pgsql/12/data logrotate
server signaled to rotate log file
```

1.6.14 OUTIL PG_CHECKSUMS

- `pg_verify_checksums` renommée en `pg_checksums`
- Activation, désactivation des *checksums*

La commande `pg_verify_checksums` (ajoutée en version 11) est renommée en `pg_checksums`. Cette commande permet de vérifier l'intégrité de la totalité des fichiers de l'instance PostgreSQL.

À partir de la version 12, cette commande possède également les arguments `--enable` et `--disable` permettant d'activer ou de désactiver les sommes de contrôle dans l'instance. Jusqu'en version 11, il était impossible de changer ce paramètre sans récréer une nouvelle instance.

Attention, l'instance doit être arrêtée pour toutes actions de la commande `pg_checksums`.

Exemple :

```
postgres@workshop12:~/12/data$ pg_checksums
pg_checksums: error: cluster must be shut down

postgres@workshop12:~/12/data$ pg_ctl -D /opt/pgsql/12/data/ stop
waiting for server to shut down.... done
server stopped

postgres@workshop12:~/12/data$ pg_checksums
Checksum operation completed
Files scanned:  1563
Blocks scanned: 4759
Bad checksums:  0
Data checksum version: 1

postgres@workshop12:~/12/data$ pg_checksums --disable
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums disabled in cluster

postgres@workshop12:~/12/data$ pg_checksums
pg_checksums: error: data checksums are not enabled in cluster

postgres@workshop12:~/12/data$ pg_checksums --enable
Checksum operation completed
Files scanned:  1563
Blocks scanned: 4759
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster
```

1.6.15 PARAMÈTRES DE CONFIGURATION

- Nouveaux paramètres
- Disparition du fichier `recovery.conf`
- Paramètres modifiés
- Paramètres dont la valeur par défaut a été modifiée

Nouveaux paramètres

> Les paramètres par défaut sont entre parenthèses.

Paramètre	Commentaire	Contexte
archive_cleanup_command	Anciennement dans le fichier recovery.conf	sighup
default_table_access_method (heap)	Spécifie la méthode d'accès aux tables par défaut à utiliser lors de la création de tables.	
log_transaction_sample_rate (0)		superuser
plan_cache_mode (auto)	Change le comportement du cache des plans d'exécution (`auto`, `force_custom_plan`, `force_generic_plan`)	user
primary_conninfo	Anciennement dans le fichier recovery.conf	postmaster
primary_slot_name	Anciennement dans le fichier recovery.conf	postmaster
promote_trigger_file	Anciennement dans le fichier recovery.conf	sighup
recovery_end_command	Anciennement dans le fichier recovery.conf	sighup
recovery_min_apply_delay (0)	Anciennement dans le fichier recovery.conf	sighup
recovery_target	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_action (pause)	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_inclusive (on)	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_lsn	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_name	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_time	Anciennement dans le fichier recovery.conf	postmaster

Paramètre	Commentaire	Contexte
recovery_target_timeline (latest)	Anciennement dans le fichier recovery.conf	postmaster
recovery_target_xid	Anciennement dans le fichier recovery.conf	postmaster
restore_comand	Anciennement dans le fichier recovery.conf	postmaster
shared_memory_type	Type de mémoire partagée, les valeurs possibles dépendent du système d'exploitation	postmaster
ssl_library	Nom de la librairie fournissant les fonctions SSL	internal
ssl_max_protocol_version	Version max du protocole SSL supporté	sighup
ssl_min_protocol_version (TLSv1)	Version min du protocole SSL supporté	sighup
tcp_user_timeout (0)		user
wal_init_zero (on)	Remplit les nouveaux fichiers WAL de zéros	superuser
wal_recycle (on)	Recycle les WAL	superuser

Paramètres modifiés

Paramètre	Changements
autovacuum_vacuum_cost_delay	Le type change de *integer* à *real*
default_with_oids	Existe toujours mais ne peut pas être à « *on* » (suppression des OID)
dynamic_shared_memory_type	Option « none » supprimée
log_autovacuum_min_duration	Contenu du journal applicatif change en fonction de l'exécution du vacuum
log_connections	L'« application_name » est ajoutée dans les lignes du journal applicatif
recovery_target_timeline	L'option « current » a été ajoutée et la nouvelle valeur par défaut est « latest »
vacuum_cost_delay	Le type change de integer à real
wal_level	Le démarrage de l'instance vérifie si le paramètre wal_level est bien renseigné
wal_sender_timeout	Contexte de modification passe de « sighup » à « user »

Paramètres ayant une nouvelle valeur par défaut.

Paramètre	PostgreSQL 11	PostgreSQL 12
autovacuum_vacuum_cost_delay	20	2
extra_float_digits	0	1
jit	OFF	ON
recovery_target_timeline		latest
transaction_isolation	default	read committed

1.7 PERFORMANCES

- `REINDEX CONCURRENTLY`
- `CREATE STATISTICS` pour les distributions non-uniformes
- Paramètre `plan_cache_mode`
- Fonctions d'appui : pour améliorer les estimations de coût des fonctions
- *JIT* par défaut
- Optimisation *CTE* : `MATERIALIZED` / `NOT MATERIALIZED`
- Meilleures performances sur le partitionnement

1.7.1 REINDEX CONCURRENTLY

- `REINDEX CONCURRENTLY`

La commande `REINDEX` peut être maintenant suivie de l'option `CONCURRENTLY`, afin de permettre la réindexation d'un index en parallèle de son utilisation.

`REINDEX CONCURRENTLY` crée un nouvel index en concurrence de l'activité usuelle sur l'ancien. Une fois le nouvel index créé et validé, il remplace alors l'ancien. C'est seulement lors de cette dernière phase très rapide que la commande nécessite un verrou exclusif sur l'index.

Exemple :

```
pg12=$ REINDEX (VERBOSE) TABLE CONCURRENTLY t1;
psql: INFO:  index "public.idx_t1_id" was reindexed
psql: INFO:  table "public.t1" was reindexed
DETAIL:  CPU: user: 1.97 s, system: 0.71 s, elapsed: 3.48 s.
REINDEX
```

L'option existe également pour la commande shell `reindexdb` (`--concurrently`).

Exemple :

```
$ reindexdb --dbname pg12 --concurrently --table t1 --verbose
INFO:  index "public.idx_t1_id" was reindexed
INFO:  table "public.t1" was reindexed
DETAIL:  CPU: user: 1.74 s, system: 0.69 s, elapsed: 3.23 s.
```

Notez qu'un `REINDEX` classique est plus rapide sans l'option `CONCURRENTLY`, ce dernier effectuant moins de travail:

```
$ pgbench -i -s 100
```

```
$ time -f%E reindexdb -i pgbench_accounts_pkey
0:06.83

$ time -f%E reindexdb --concurrently -i pgbench_accounts_pkey
0:09.34
```

Néanmoins, la reconstruction se faisant en concurrence avec la production usuelle, ce temps supplémentaire est moins impactant. Voici un exemple:

```
#### SANS CONCURRENTLY

$ pgbench -c1 -T20 -S & reindexdb -i pgbench_accounts_pkey
[...]
number of transactions actually processed: 66241
latency average = 0.302 ms
tps = 3312.017682 (including connections establishing)
tps = 3312.738804 (excluding connections establishing)

#### AVEC CONCURRENTLY

$ pgbench -c1 -T20 -S & reindexdb --concurrently -i pgbench_accounts_pkey
[...]
number of transactions actually processed: 118991
latency average = 0.168 ms
tps = 5949.500860 (including connections establishing)
tps = 5951.142968 (excluding connections establishing)
```

Le nombre de transactions par seconde est plus important avec l'option `CONCURRENTLY` (5951 tps contre 3312), ce dernier n'ayant bloqué la production qu'un court instant.

1.7.2 CREATE STATISTICS MCV

- Nouveau type `MCV` pour la commande `CREATE STATISTICS`
- *Most Common Values*
- Collecte les valeurs les plus communes pour un ensemble de colonnes

Jusqu'en version 11, la commande `CREATE STATISTICS` supportait deux types de collecte de statistique: `n-distinct` et `dependencies`.

Le type `mcv` ajouté en version 12 permet de créer des statistiques sur les valeurs les plus communes pour les colonnes indiquées.

Les statistiques sont stockées dans la table `pg_statistic_ext`.

Exemple :

```
pg12=$ CREATE TABLE t4 (id INT, nb NUMERIC, comm varchar(50));
CREATE TABLE

pg12=$ CREATE STATISTICS stat_mcv_t4(mcv) ON id, nb FROM t4 ;
CREATE STATISTICS

pg12=$ select * from pg_statistic_ext;
  oid  | stxrelid |   stxname   | stxnamespace | stxowner | stxkeys | stxkind
-------+----------+-------------+--------------+----------+---------+---------
 41010 |    41003 | stat_mcv_t4 |         2200 |       10 | 1 2     | {m}
(1 row)
```

Pour plus d'information et d'exemple, voir le chapitre Extended Statistics[10]

1.7.3 MISE EN CACHE DES PLANS D'EXÉCUTION

- `plan_cache_mode`
- Trois modes:
  - `auto`
  - `force_custom_plan`
  - `force_generic_plan`

Ce nouveau paramètre permet de définir la méthode de mise en cache du plan d'exécution des instructions préparées (eg. commande `PREPARE`).

La valeur par défaut est `auto`, ce qui correspond au comportement habituel du moteur: utiliser le plan générique si son coût n'est pas beaucoup plus important que celui des cinq premières exécutions réalisées de la requête.

Les deux autres valeurs `force_custom_plan` et `force_generic_plan` permettent respectivement de forcer l'utilisation d'un plan calculé à chaque exécution, ou au contraire de forcer l'utilisation d'un plan générique.

Note : Le paramètre est appliqué lorsqu'un plan mis en cache doit être exécuté, pas lorsqu'il est préparé.

---

[10] https://www.postgresql.org/docs/12/planner-stats.html#PLANNER-STATS-EXTENDED

1.7.4 FONCTIONS D'APPUI (SUPPORT FUNCTIONS)

- Améliore la visibilité du planificateur sur les fonctions
- Possibilité d'associer à une fonction une fonction « de support »
- Produit dynamiquement des informations sur:
  - la sélectivité
  - le nombre de lignes produit
  - son coût d'exécution
- La fonction doit être écrite en C

Jusqu'en version 11, le planificateur considérait des fonctions comme des boîtes noires, avec éventuellement quelques informations très partielles et surtout statiques à propos de leur coût et du nombre de lignes retourné.

Les *supports functions* permettent de fournir dynamiquement des informations au planificateurs concernant les fonctions utilisées et leur résultat dans le contexte de la requête.

Voici un exemple avec la fonction `generate_series`:

```
# \sf generate_series(int, int)
CREATE OR REPLACE FUNCTION pg_catalog.generate_series(integer, integer)
 RETURNS SETOF integer
 LANGUAGE internal
 IMMUTABLE PARALLEL SAFE STRICT SUPPORT generate_series_int4_support
AS $function$generate_series_int4$function$
```

```
# explain select i from generate_series(1,10000) t(i);
                                QUERY PLAN
--------------------------------------------------------------------------
 Function Scan on generate_series t  (cost=0.00..100.00 rows=10000 width=4)
```

```
# explain select i from generate_series(1,10) t(i);
                            QUERY PLAN
-------------------------------------------------------------------
 Function Scan on generate_series t  (cost=0.00..0.10 rows=10 width=4)
```

```
# explain select i from generate_series(1,10) t(i) where i > 9;
                            QUERY PLAN
-------------------------------------------------------------------
```

```
Function Scan on generate_series t  (cost=0.00..0.13 rows=3 width=4)
  Filter: (i > 9)
```

1.7.5 JIT PAR DÉFAUT

- JIT (Just-In-time) est maintenant activé par défaut

La compilation *JIT* (*Just-In-time*) est maintenant active par défaut dans PostgreSQL 12.

Les informations de compilation *JIT*, peuvent être loggées dans le journal.

1.7.6 MODIFICATION DU COMPORTEMENT PAR DÉFAUT DES REQUÊTES CTE

- Les *CTE* ne sont plus par défaut des barrières d'optimisation
- Modification du comportement par défaut des CTE
- Nouvelles syntaxes:
  - `MATERIALIZED`
  - `NOT MATERIALIZED`

Jusqu'en version 11, les *CTE* (*Common Table Expression*) spécifiées dans la clause `WITH` étaient "matérialisées". Autrement dit, les CTE devaient être exécutés tels quels sans optimisation possible avec le reste de la requête. C'est ce qu'on appelle une « barrière d'optimisation »

Depuis la version 12, ce comportement n'est plus le même. Par défaut, les *CTE* ne sont plus des barrières d'optimisation, ce qui permet de déplacer certaines opérations de la requête afin de les rendre plus efficaces.

Il est possible de forcer l'un ou l'autre des comportements grâce aux syntaxes `MATERIALIZED` ou `NOT MATERIALIZED` de la clause `WITH`.

Par exemple, le fait de spécifier l'option `NOT MATERIALIZED`, permet à la clause `WHERE` de pousser les restrictions à l'intérieur de la clause `WITH`.

Les conditions des requêtes pour l'application de l'option `NOT MATERIALIZED` sont :

- requêtes non récursives
- requêtes référencées une seule fois
- requêtes n'ayant aucun effet de bord

Exemple : dans cet exemple, on voit que l'index sur la colonne `id` n'est pas utilisé à cause du CTE.

```
pg12=$ EXPLAIN ANALYZE WITH rq AS MATERIALIZED (SELECT * FROM t1)
SELECT * FROM rq WHERE id=1500;

                                        QUERY PLAN
-------------------------------------------------------------------------------------
 CTE Scan on rq  (cost=133470.68..201273.71 rows=15067 width=4)
(actual time=19.185..7067.369 rows=4 loops=1)
   Filter: (id = 1500)
   Rows Removed by Filter: 3100096
   CTE rq
     ->  Seq Scan on t1  (cost=0.00..133470.68 rows=3013468 width=4)
(actual time=4.311..3749.203 rows=3100100 loops=1)
 Planning Time: 0.195 ms
 Execution Time: 7073.119 ms
```

La même requête avec l'option `NOT MATERIALIZED` (valeur par défaut si l'option n'est pas spécifiée) permet de réduire le temps d'exécution.

```
pg12=$ EXPLAIN ANALYSE WITH rq AS NOT MATERIALIZED (SELECT * FROM t1)
SELECT * FROM rq WHERE id =1500;
                                        QUERY PLAN
-------------------------------------------------------------------------------------
 Index Only Scan using idx_t1_id on t1  (cost=0.43..15.26 rows=3 width=4)
(actual time=0.184..0.245 rows=4 loops=1)
   Index Cond: (id = 1500)
   Heap Fetches: 0
 Planning Time: 0.209 ms
 Execution Time: 0.338 ms
```

1.7.7 PERFORMANCES DU PARTITIONNEMENT

- Performances accrues pour les tables avec un grand nombre de partitions
- Verrous lors des manipulations de partitions
- Support des clés étrangères vers une table partitionnée
- Amélioration du chargement de données

PostgreSQL 12 optimise l'accès aux tables ayant plusieurs milliers de partitions.

La commande `ATTACH PARTITION`, ne verrouille plus de façon exclusive la table partitionnée, permettant ainsi de ne pas bloquer la production lors de l'ajout d'une partition. En revanche, l'option `DETACH PARTITION`, pose toujours un verrou exclusif.

L'utilisation de clés étrangères pour les partitions filles est désormais supportée. Voir à ce propos le chapitre sur le partitionnement.

Une amélioration de la fonction `COPY` dans les partitions permet un chargement plus rapide.

1.8 INCOMPATIBILITÉS

- Disparition du `recovery.conf`
- `max_wal_senders` plus inclus dans `max_connections`
- Noms des clés étrangères
- Tables `WITH OIDS` n'existent plus
- Types de données supprimés
- Fonctions `to_timestamp` et `to_date`
- Outil `pg_checksums`

1.8.1 DISPARITION DU FICHIER RECOVERY.CONF.

- Fichier `recovery.conf` fusionné avec les paramètres normaux

Les paramètres de l'ancien `recovery.conf` deviennent des `GUC` (*Grand Unified Configuration*) comme tous les autres paramètres. À ce titre, ils peuvent donc être positionnés dans le fichier `postgresql.conf` ou tout autres moyens (au démarrage, commande `ALTER SYSTEM`, fichier inclus, etc). L'ancien fichier `recovery.conf` quant à lui disparaît.

En conséquence, la présence d'un fichier `recovery.conf` dans le répertoire racine d'une instance PostgreSQL 12 bloque son démarrage.

Attention, les scripts et outils de gestion des PITR (eg. générant automatiquement le fichier `recovery.conf`) doivent donc être mis à mettre à jour en même temps que l'instance PostgreSQL 12.

De même, la gestion d'instance secondaire en réplication est aussi impactée. Tout l'outillage mis en œuvre doit être mis à jour.

1.8.2 `MAX_WAL_SENDERS` N'EST PLUS INCLUS DANS `MAX_CONNECTIONS`

- Les `wal senders` ne sont plus comptabilisés dans `max_connections`
- Impact potentiel sur les seuils de supervision
- Impact plus anecdotique pour ces paramètres

Les processus *WAL senders* sont chargés d'envoyer le contenu des WAL aux instances en réplication. Chaque instance secondaire maintient une connexion sur l'instance de production au travers de son wal sender attitré.

Depuis la version 12, le nombre de *wal senders* n'est plus décompté du nombre maximum de connexions (`max_connections`) autorisé.

Si des outils de supervision prenaient en compte le calcul (`max_connections - max_wal_senders` par exemple), les sondes peuvent renvoyer des informations légèrement faussées.

1.8.3 NOMS DES CLÉS ÉTRANGÈRES AUTOGÉNÉRÉES

- Changement de norme de nommage des FK
- Inclut désormais toutes les colonnes concernées
- Impact potentiel sur les outils

La génération automatique des noms de clés étrangères, prend désormais en compte le nom de toutes les colonnes de la clé. Cela peut entraîner une incompatibilité dans certains scripts qui se basent sur le nom des clés étrangères générées par PostgreSQL avec une colonne.

Exemple :

```
pg12=> CREATE TABLE t2(id INT, id2 INT, comm VARCHAR(50),
FOREIGN KEY (id, id2) REFERENCES t1(id, id2)) ;
CREATE TABLE
postgres=> \d t2
Table "public.t2"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 id     | integer               |           |          |
 id2    | integer               |           |          |
 comm   | character varying(50) |           |          |
Foreign-key constraints:
    "t2_id_id2_fkey" FOREIGN KEY (id, id2) REFERENCES t1(id, id2)
```

1.8.4 WITH OIDS

- Attribut `WITH OIDS` supprimé en version 12
- Colonnes `oid` « système » deviennent visibles

L'option `WITH OIDS` (instruction `CREATE TABLE`) est supprimée. Il n'est plus possible de créer des tables avec une colonne `oid` « cachée ».

L'option `WITHOUT OIDS` est toujours supportée, et le paramètre `default_with_oids` n'existe qu'en lecture seule avec comme valeur `off`.

Exemple : impossible de créer une table avec WITH OIDS

```
pg12=#  CREATE TABLE t3(id INT) WITH OIDS ;
psql: ERROR:  syntax error at or near "OIDS"
LINE 1: CREATE TABLE t3(id INT) WITH OIDS ;
```

Si cette colonne vous est utile, il faudra la créer explicitement. Elle apparaît alors auprès des autres colonnes lors de requêtes de type `SELECT * FROM ...` ou `TABLE ...`.

Coté catalogue système, toutes les colonnes `oid` deviennent visibles. Un `SELECT *` sur ces tables affiche donc la colonne supplémentaire.

Exemple : la table pg_class à une colonne `oid` visible

```
pg12=#  postgres=# \d pg_class
                    Table "pg_catalog.pg_class"
       Column        |     Type     | Collation | Nullable | Default
---------------------+--------------+-----------+----------+---------
 oid                 | oid          |           | not null |
 relname             | name         |           | not null |
 relnamespace        | oid          |           | not null |
 reltype             | oid          |           | not null |
 .......
```

L'option `-o` ou `--oids` de la commande `pg_dump` a également été supprimée.

1.8.5 TYPE DE DONNÉES SUPPRIMÉS

Suppression des types suivants:

- `abstime`
- `reltime`
- `tinterval`

Les types de données suivant ont été supprimés:

- `abstime`
- `reltime`
- `tinterval`

L'utilisation de ces types était explicitement découragée et dépréciée dans la documentation depuis la version…7.0, en l'an 2000.

Ces types peuvent être avantageusement remplacés par les types `timestamp` et `interval`, ou leurs dérivés.

Le type `abstime` de la colonne `valuntil` de la vue `pg_shadow` a été en conséquence remplacé par `timestamp with time zone`.

1.8.6 FONCTIONS TO_TIMESTAMP ET TO_DATE

- Correction des fonctions `to_timestamp` et `to_date`
- Changement de comportement

Les espaces inutiles sont supprimés dans les modèles de formatage des fonctions `to_timestamp` et `to_date`.

Exemple :

Jusqu'en version 11, cet appel de fonction renvoie une erreur ou un résultat faux :

```
pg11=> SELECT TO_DATE('2019/08/25', ' YYYY/MM/DD') ;
  to_date
------------
0019-08-25

pg11=> SELECT TO_DATE('2019/08/25', ' YYYY/  MM/DD') ;
ERREUR:  valeur « /2 » invalide pour « MM »
DÉTAIL : La valeur doit être un entier
```

Dans PostgreSQL 12, les espaces inutiles sont supprimés et ignorés:

```
pg12=> SELECT TO_DATE('2019/08/25', ' YYYY/MM/DD') ;
  to_date
------------
2019-08-25

pg12=> SELECT TO_DATE('2019/08/25', ' YYYY/  MM/DD') ;
  to_date
------------
2019-08-25
```

Ce comportement peut potentiellement (quoique très rarement) avoir un impact sur la couche applicative.

1.8.7 PG_VERIFY_CHECKSUMS RENOMMÉE EN PG_CHECKSUMS

L'outil pg_verify_checksums devient pg_checksums

La fonction pg_verify_checksums n'existe plus, elle est remplacée par pg_checksums avec les mêmes fonctionnalités.

Attention à vos scripts de supervision ou de vérification !

1.9 FONCTIONNALITÉS FUTURES

- *Pluggable storage*
  - *HEAP storage*, historique et par défaut
  - *columnar storage*
  - *Zed Heap*
  - *blackhole*

Jusqu'à lors, le stockage des données s'effectuait dans les tables à l'aide d'un mécanisme appellé *Heap Storage*. Cette méthode était l'unique implémentation du stockage dans PostgreSQL pour le contenu des tables ou des vues matérialisées.

Avec la version 12, l'ajout des *pluggable storages* apporte une nouvelle couche d'abstraction dans la gestion des accès aux données des tables et des vues.

D'autres méthodes de stockage sont en cours de développement et permettront de choisir un format en fonction de l'utilisation des données stockées, table par table. Cette amélioration permettra de répondre à des attentes utilisateurs déjà présentes dans les autres moteurs SGBD du marché.

Pour aller plus loin : Présentation d'Andres Freund PGConf-EU 2018[11]

1.9.1 PLUGGABLE STORAGE

1.9.1.1 HEAP storage

- `HEAP storage`
- Méthode de stockage par défaut
- Seule méthode supportée pour le moment

La méthode de stockage traditionnelle de PostgreSQL pour ses objets, a été adaptée en tant qu'*Access Method* utilisant la nouvelle architecture. Cette méthode a été simplement appelée `HEAP` et est pour le moment la seule supportée.

---

[11] https://anarazel.de/talks/2018-10-25-pgconfeu-pluggable-storage/pluggable.pdf

1.9.1.2 Méthode d'accès ZedStore (Columnar storage)

- Méthode orientée colonne
- Données compressées
- Nom temporaire !

Heikki Linnakangas de Pivotal travaille sur le ***columnar storage***, une méthode de stockage orientée « colonne » et permettant entre autres, la compression des données des colonnes.

Bénéfices :

- Optimisation en cas de nombreuses mises à jour sur une même colonne ;
- Suppression de colonne instantanée ;
- Possibilité de réécrire une colonne plutôt que toute la table.

Pour plus d'informations sur cette méthode de stockage et ses développements, voir les slides de conférence[12] de Heikki Linnakangas à ce sujet.

1.9.1.3 Méthode d'accès zHeap

- Meilleur contrôle du *bloat*
- Réduction de l'amplification des écritures
- Réduction de la taille des entêtes
- Méthode basée sur les différences

EnterpriseDB travaille actuellement sur une méthode de stockage nommée *zHeap* dont le fonctionnement repose sur un système UNDO en lieu et place du REDO actuel. Le principe est de modifier les enregistrements "sur place" lorsque c'est possible et de conserver dans les journaux de transaction l'information suffisante pour retourner à l'état précédent en cas de ROLLBACK.

Les bénéfices observés seraient :

- un meilleur contrôle du *bloat*
- la réduction de l'amplification des écritures comparativement à la méthode `HEAP`
- stockage plus performant en réduisant la taille des entêtes

Pour aller plus loin : Article du contributeur Amit Kapila d'EntrepriseDB[13]

---

[12] https://www.postgresql.eu/events/pgconfeu2019/schedule/session/2738-zedstore-column-store-for-postgresql/
[13] https://www.enterprisedb.com/blog/zheap-storage-engine-provide-better-control-over-bloat

1.9.1.4 Méthode d'accès *Blackhole*

- Sert de base pour créer une extension *Access Method*
- Toute donnée ajoutée est envoyée dans le néant

Cette extension écrite par Michael Paquier[14] , fournit une base pour l'écriture des extensions pour les méthodes d'accès. Toutes les données sont envoyées dans le néant.

```
$ CREATE EXTENSION blackhole_am;
CREATE EXTENSION
$ \dx+ blackhole_am
   Objects in extension "blackhole_am"
            Object description
------------------------------------------
 access method blackhole_am
 function blackhole_am_handler(internal)
(2 rows)

$ CREATE TABLE blackhole_tab (id int) USING blackhole_am;
CREATE TABLE
$ INSERT INTO blackhole_tab VALUES (generate_series(1,100));
INSERT 0 100
$ SELECT * FROM blackhole_tab;
 id
----
(0 rows)
```

[14] https://github.com/michaelpq/pg_plugins/tree/master/blackhole_am

2 ATELIERS

2.1 TP SUR LES COLONNES GÉNÉRÉES

- Création d'une colonne générée
- Observations lors d'un `UPDATE`
- Utilisation des index

2.1.1 DÉFINITION DE LA TABLE

Pour ce rapide travail pratique, nous créons une table `table1` avec une colonne générée à partir de deux autres colonnes.

```
CREATE TABLE table1(
  id serial PRIMARY KEY,
  a int NOT NULL DEFAULT 0,
  b int NOT NULL DEFAULT 0,
  prod int generated always as (a * b) stored
);
```

On constate la définition `generated always as stored` dans la description de la colonne `prod` pour notre table.

```
$ \d table1
Table "public.table1"
Column |  Type   | Collation | Nullable |              Default
-------+---------+-----------+----------+-----------------------------------
id     | integer |           | not null | nextval('table1_id_seq'::regclass)
a      | integer |           | not null | 0
b      | integer |           | not null | 0
prod   | integer |           |          | generated always as (a * b) stored
Indexes:
  "table1_pkey" PRIMARY KEY, btree (id)
```

2.1.2 MODIFICATIONS DU CONTENU DE LA TABLE

À l'ajout d'une nouvelle ligne dans la table `table1`, le moteur génére automatiquement la valeur `prod` calculée à partir des valeurs `a` et `b`.

```
INSERT INTO table1 (a,b) VALUES (6,7);
SELECT * FROM table1;

 id | a | b | prod
----+---+---+------
  1 | 6 | 7 |   42
(1 row)
```

Lors d'une modification, le fonctionnement interne MVCC crée une nouvelle version de la ligne et recalcule à la volée la valeur `prod`. À la validation de la transaction (`COMMIT`), toutes les nouvelles transactions verront la nouvelle version de la ligne et la valeur de la colonne générée `prod`.

```
UPDATE table1 SET a=7 WHERE a=6;
SELECT * FROM table1;

 id | a | b | prod
----+---+---+------
  1 | 7 | 7 |   49
(1 row)
```

2.1.3 TENTATIVE DE MODIFICATION DE LA COLONNE GÉNÉRÉE

Les colonnes générées sont en lecture seule et ne peuvent être modifiées que lors d'une réévaluation à l'écriture de la ligne (`INSERT` ou `UPDATE`). Les modifications des colonnes calculées sont interdites à moins de rétablir la valeur par défaut :

```
UPDATE table1 SET prod = 43;

ERROR:  column "prod" can only be updated to DEFAULT
DETAIL:  Column "prod" is a generated column.
```

2.1.4 AJOUT D'UN INDEX SUR UNE COLONNNE GÉNÉRÉE

Puisque la donnée d'une colonne générée est stockée (`STORED`) aux côtés des données de la table, le mécanisme d'indexation est tout à fait valable.

Nous pouvons alimenter la table `table1` avec un peu plus de données avant de créer un index sur la colonne `prod`.

```
TRUNCATE TABLE table1;
INSERT INTO table1 (a,b)
  SELECT (random()*100)::int+i, (random()*10)::int+i
    FROM generate_series(1,10000) as i;

CREATE INDEX ON table1(prod);
```

L'index sera utilisé lors d'une recherche sur la colonne `prod` comme le montre le plan d'exécution suivant (`Index Scan using table1_prod_idx on table1`) :

```
EXPLAIN (analyze,buffers)
  SELECT a,b,prod FROM table1 WHERE prod BETWEEN 10 AND 100;

                        QUERY PLAN
-----------------------------------------------------------------
 Index Scan using table1_prod_idx on table1
  (cost=0.29..8.30 rows=1 width=12)
  (actual time=0.011..0.013 rows=1 loops=1)
   Index Cond: ((prod >= 10) AND (prod <= 100))
   Buffers: shared hit=3
 Planning Time: 0.183 ms
 Execution Time: 0.036 ms
(5 rows)
```

2.2 TP SUR LE PARTITIONNEMENT

- Transformation d'une table référencée en table partitionnée
- Fonctions d'information sur le partitionnement
- Nouvelle commande `\dP` pour les partitions

2.2.1 CLÉS ÉTRANGÈRES VERS UNE TABLE PARTITIONNÉE

Créer les deux tables `job` et `job_detail` avec une contrainte de clé étrangère.

```
CREATE TABLE job (
   id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   job_start TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
   job_end TIMESTAMP WITH TIME ZONE,
   job_name VARCHAR(50) NOT NULL
);
CREATE INDEX ON job(job_start);

CREATE TABLE job_detail (
   jobid INT REFERENCES job(id) ON DELETE CASCADE NOT NULL,
   log_date TIMESTAMP WITH TIME ZONE NOT NULL,
   log_message TEXT NOT NULL
);

-- Quelques données dans les tables
INSERT INTO job (job_end, job_name) VALUES
   (now() + INTERVAL '1 hour', 'Daily routine');
INSERT INTO job_detail VALUES
   (currval('job_id_seq'), now(), 'Purge is started');
INSERT INTO job_detail VALUES
   (currval('job_id_seq'), now() + INTERVAL '1 hour', 'Purge is completed');

INSERT INTO job (job_end, job_name) VALUES
   (now() + INTERVAL '2 hour', 'Other daily routine');
INSERT INTO job_detail VALUES
   (currval('job_id_seq'), now(), 'Routine is started');
INSERT INTO job_detail VALUES
   (currval('job_id_seq'), now() + INTERVAL '2 hour', 'Routine is completed');
```

Dans un contexte de croissance, le nombre de travaux (*jobs*) augmente considérablement et la purge de la table devient périlleuse au long terme. La transformation de cette table en table partionnée se revèle nécessaire.

Créer une table `job_part` ayant la même structure que la table `job` en ajoutant le champ `job_start` dans la contrainte de clé primaire pour satisfaire les prérequis de la clé de

partionnement.

```
CREATE TABLE job_part
   (LIKE job INCLUDING DEFAULTS INCLUDING IDENTITY )
   PARTITION BY RANGE (job_start);
```

La prochaine étape consiste à rattacher la table `job` en tant que partition par défaut de la future table partitionnée. Pour cela, il peut être judicieux de jouer les instructions dans une seule transaction et de prêter garde au respect de la contrainte étrangère sur la table `job_detail`.

```
START TRANSACTION;

-- Redéfinition de la contrainte de clé primaire sur la table partitionnée
ALTER TABLE job_detail DROP CONSTRAINT IF EXISTS job_detail_jobid_fkey;
ALTER TABLE job DROP CONSTRAINT IF EXISTS job_pkey;

-- Ajout de la table job en tant que partition
ALTER TABLE job_part ATTACH PARTITION job DEFAULT;
ALTER TABLE job_PART ADD PRIMARY KEY (id, job_start);
CREATE INDEX ON job_part(job_start);

-- Récupération de la dernière valeur de la séquence de la précédente table
SELECT setval(pg_get_serial_sequence('job_part', 'id'),
       nextval(pg_get_serial_sequence('job', 'id')), true);

-- Suppression de l'ancienne séquence et de sa relation avec la partition job
ALTER TABLE job ALTER id DROP IDENTITY, ALTER job_start DROP DEFAULT;

-- Réactivation des contraintes de clés étrangères
-- Puisque la colonne job_start fait à présent partie de la PRIMARY KEY, il est
-- nécessaire d'ajouter cette colonne dans la table job_detail et de l'alimenter
ALTER TABLE job_detail ADD job_start TIMESTAMP WITH TIME ZONE;
UPDATE job_detail SET job_start = j.job_start
  FROM job j WHERE (j.id = jobid) ;

ALTER TABLE job_detail ADD FOREIGN KEY (jobid, job_start)
   REFERENCES job(id, job_start) ON DELETE CASCADE;

COMMIT;
```

Une étape optionnelle serait de renommer les relations pour être totalement transparent avec le fonctionnement applicatif.

```
START TRANSACTION;

-- Renommage des relations pour maintenir la logique métier
ALTER TABLE job RENAME TO job_default;
```

```
ALTER TABLE job_part RENAME TO job;
ALTER SEQUENCE job_part_id_seq RENAME TO job_id_seq;

COMMIT;
```

À partir de cette étape, la table `job` est partitionnée mais toutes les lignes passées et à venir sont stockées dans la partition `job_default`. Il est possible de déplacer les lignes d'une partition vers une nouvelle, bien qu'il soit recommandé de maîtriser les requêtes en provenance des utilisateurs car une série de verrous seront posés entre les tables.

```
CREATE OR REPLACE PROCEDURE add_daily_partition(timestamp with time zone)
   LANGUAGE plpgsql AS $$
   DECLARE
      daily_interval INTERVAL := INTERVAL '1 day';
      tablename    VARCHAR(50) := format('job_%s', TO_CHAR($1, 'YYYYMMDD'));
      from_expr    VARCHAR(50) := date_trunc('day', $1);
      to_expr      VARCHAR(50) := date_trunc('day', $1 + daily_interval);

   BEGIN
      EXECUTE format(
         'CREATE TABLE IF NOT EXISTS %s (LIKE job_default INCLUDING CONSTRAINTS);',
         tablename
      );
      EXECUTE format(
         'INSERT INTO %s SELECT * FROM job_default WHERE job_start BETWEEN ''%s'' AND ''%s'';',
         tablename, from_expr, to_expr
      );
      EXECUTE format(
         'DELETE FROM job_default WHERE job_start BETWEEN ''%s'' AND ''%s'';',
         from_expr, to_expr
      );
      EXECUTE format(
         'ALTER TABLE job ATTACH PARTITION %s FOR VALUES FROM (''%s'') TO (''%s'');',
         tablename, from_expr, to_expr
      );
   END;
$$;

START TRANSACTION;
LOCK TABLE job;
LOCK TABLE job_detail;

-- Retrait temporaire de la clé étrangère pour réduire les risques
-- de suppression en cascade
ALTER TABLE job_detail DROP CONSTRAINT IF EXISTS job_detail_jobid_job_start_fkey;
CALL add_daily_partition (now());
ALTER TABLE job_detail ADD FOREIGN KEY (jobid, job_start)
```

```
   REFERENCES job(id, job_start) ON DELETE CASCADE;

COMMIT;
```

2.2.2 FONCTIONS D'INFORMATION SUR LE PARTITIONNEMENT

`pg_partition_root` renvoie la partition mère d'une partition.

```
$ SELECT pg_partition_root('job_default');

 pg_partition_root
-------------------
 job
(1 row)
```

`pg_partition_ancestors` renvoie la partition mère ainsi que la partition concernée.

```
$ SELECT pg_partition_ancestors('job_default');

 pg_partition_ancestors
------------------------
 job_default
 job
(2 rows)
```

`pg_partition_tree` renvoie tout l'arbre de la partition sous forme de tuples.

```
$ SELECT * from pg_partition_tree('job');

    relid     | parentrelid | isleaf | level
--------------+-------------+--------+-------
 job          |             | f      |     0
 job_default  | job         | t      |     1
 job_20200124 | job         | t      |     1
(3 rows)
```

2.2.3 AFFICHAGE DES TABLES PARTITIONNÉES SEULES

La commande psql `\dP` permet à présent d'afficher les tables partitionnées, contrairement à la commande `\d` qui affiche toutes les relations de la base.

```
$ \d
                 List of relations
 Schema |     Name     |       Type        |  Owner
--------+--------------+-------------------+----------
 public | job          | partitioned table | postgres
 public | job_20200124 | table             | postgres
```

```
 public | job_default  | table             | postgres
 public | job_detail   | table             | postgres
 public | job_id_seq   | sequence          | postgres
(5 rows)

$ \dP
                 List of partitioned relations
 Schema |       Name         |  Owner   |       Type        | Table
--------+--------------------+----------+-------------------+-------
 public | job                | postgres | partitioned table |
 public | job_job_start_idx  | postgres | partitioned index | job
 public | job_part_pkey      | postgres | partitioned index | job
(3 rows)
```

2.3 TP MONITORING

- Échantillon des requêtes dans les logs
- Vues de progression des opérations de maintenance
- `pg_ls_archive_statusdir()` et `pg_ls_tmpdir()`
- `pg_stat_replication` : timestamp du dernier message reçu du standby

2.3.1 ÉCHANTILLON DES REQUÊTES DANS LES LOGS

Les deux paramètres sont à modifier dans le fichier `postgresql.conf` de l'instance :

```
# on récupère toute l'activité
log_min_duration_statement = 0

# 80% des transactions
log_transaction_sample_rate = 0.80
```

2.3.2 VUES DE PROGRESSION DES OPÉRATIONS DE MAINTENANCE

2.3.2.1 Surveiller la progression d'une création d'index

Dans une première session `psql`, lancer les commandes suivante pour observer les opérations de maintenance sur index avec la vue `pg_stat_progress_create_index` :

```
$ SELECT datname, relid::regclass, command, phase, tuples_done
    FROM pg_stat_progress_create_index;

$ \watch 1
```

Dans une autre session, alimenter une table puis créer un index :

```
$ CREATE TABLE a_table (i int, a text, b text);
$ INSERT INTO a_table SELECT i, md5(i::text), md5(i::text)
    FROM generate_series(1, 100000) i;

$ CREATE INDEX ON a_table (i, a, b);
```

Observer les phases de la création de l'index dans la première session.

2.3.2.2 Surveiller la progression d'un vacuum

On se propose de suivre la progression des vacuum d'une base. Ces derniers se déclenchent lorsqu'un certain seuil de modifications est atteint sur un ou plusieurs tables.

```
$ CREATE TABLE tab (i int, j text);
$ INSERT INTO tab SELECT i, md5(i::text)
    FROM generate_series(1,1000000) s(i);

$ DELETE FROM tab;
$ INSERT INTO tab SELECT i, md5(i::text)
    FROM generate_series(1,1000000) s(i);

-- L'autovacuum devrait se déclencher
$ SELECT datname, relid::regclass, phase, heap_blks_total, num_dead_tuples
    FROM pg_stat_progress_vacuum;
$ \watch 1
```

2.3.3 PG_LS_ARCHIVE_STATUSDIR() ET PG_LS_TMPDIR()

Déclencher la rotation des journaux de transactions et donc l'archivage :

```
$ SELECT pg_switch_wal();
$ SELECT pg_switch_wal();
$ SELECT pg_switch_wal();
$ SELECT pg_ls_archive_statusdir();

                  pg_ls_archive_statusdir
-------------------------------------------------------------
 (000000010000000100000000BC.done,0,"2019-07-29 14:12:08+02")
 (000000010000000100000000BB.done,0,"2019-07-29 14:11:36+02")
 (000000010000000100000000BD.done,0,"2019-07-29 14:12:15+02")
(3 rows)
```

Lancer une requête effectuant un tri assez conséquent pour dépasser la `work_mem` et créer des fichiers temporaires :

```
$ SELECT pg_ls_tmpdir();
                     pg_ls_tmpdir
-----------------------------------------------------
 (pgsql_tmp16064.9,24395776,"2019-07-29 14:05:49+02")
(1 row)
```

2.3.4 PG_STAT_REPLICATION : TIMESTAMP DU DERNIER MESSAGE REÇU DU STANDBY

2.3.4.1 Réplication sur le serveur primaire

Nous allons mettre en place une réplication logique entre deux instances v12.

Récupérer les sources, compiler installer dans `/usr/local/pgsql`. Créer un répertoire `12` dans le `home` de l'utilisateur et initialiser un cluster dedans :

```
$ /usr/local/pgsql/bin/initdb -D data
```

Créer deux scripts simples de démarrage et d'arrêt :

```
#! /bin/bash
# start.sh

/usr/local/pgsql/bin/pg_ctl -D data -l logfile start
```

```
#! /bin/bash
# stop.sh

kill $(cat data/postmaster.pid | head -1)
```

Activation de la réplication logique sur le primaire :

postgresql.conf

```
wal_level = logical
```

Création de l'utilisateur de réplication :

```
$ create user repli with password 'repli';
CREATE ROLE
$ alter user repli  with replication ;
$ grant select on all tables in schema public to repli;
GRANT
```

Création de la publication sur la table `a_table` :

```
$ create table a_table (i int primary key, a text,b text);
$ insert into a_table select i,i::text,i::text from generate_series (1,1000000) i ;
$ create user repli with replication;
$ CREATE publication a_table_publication for table a_table;
```

pg_hba.conf

```
host all repli 127.0.0.1/32 md5
```

Création de la structure de la table répliquée depuis le serveur primaire :

```
$ pg_dump -s -t a_table -U postgres |/usr/local/pgsql/bin/psql -p 5412 postgres
```

2.3.4.2 Souscription sur le serveur secondaire

```
$ create subscription a_table_subscription
CONNECTION 'host=127.0.0.1 port=5432 user=repli dbname=postgres password=repli'
publication a_table_publication ;
```

2.3.4.3 Vérification de la Réplication

```
$ select usename,application_name,reply_time from pg_stat_replication ;
 usename |   application_name   |          reply_time
---------+----------------------+-------------------------------
 repli   | a_table_subscription | 2019-07-30 12:14:41.934615+02
(1 row)
```

Nous avons l'heure exacte du dernier contact avec la souscription *a_table_subscription*.

2.4 TP INDEX ET PERFORMANCES

- Les fonctions d'appui
- `REINDEX CONCURRENTLY`

2.4.1 TP SUR LES FONCTIONS D'APPUI

En regardant la définition de la fonction `unnest(anyarray)`, on remarque que la v12 fait apparaître une fonction d'appui (*support function* en anglais) :

En v11

```
v11 $ \ef unnest(anyarray)

CREATE OR REPLACE FUNCTION pg_catalog.unnest(anyarray)
 RETURNS SETOF anyelement
 LANGUAGE internal
 IMMUTABLE PARALLEL SAFE STRICT ROWS 100
AS $function$array_unnest$function$
```

En v12 :

```
v12 $ \ef unnest(anyarray)

CREATE OR REPLACE FUNCTION pg_catalog.unnest(anyarray)
 RETURNS SETOF anyelement
 LANGUAGE internal
 IMMUTABLE PARALLEL SAFE STRICT ROWS 100 SUPPORT array_unnest_support
AS $function$array_unnest$function$
```

Le rôle de la fonction d'appui peut être vérifié rapidement en comparant l'`explain` sous une version 11 et sous une version 12 :

```
v11 $ explain select * from unnest(array[0,1,2,3]);
                          QUERY PLAN
---------------------------------------------------------------
 Function Scan on unnest  (cost=0.00..1.00 rows=100 width=4)
(1 row)
```

On remarque que l'optimiseur fait une erreur en estimant le nombre de lignes retournées (100 plutôt que 4).

```
v12 $ explain select * from unnest(array[0,1,2,3]);
                          QUERY PLAN
---------------------------------------------------------------
 Function Scan on unnest  (cost=0.00..0.04 rows=4 width=4)
(1 row)
```

La v12 estime bien le nombre de ligne car elle consulte la fonction d'appui.

Autre exemple avec la fonction `generate_series(bigint,bigint)`

```
v11 $ explain select generate_series(1,100000000);
                    QUERY PLAN
-----------------------------------------------
 ProjectSet  (cost=0.00..5.02 rows=1000 width=4)
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(2 rows)
```

```
v12 $ explain select generate_series(1,100000000);
                         QUERY PLAN
------------------------------------------------------------
 ProjectSet  (cost=0.00..500000.02 rows=100000000 width=4)
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(2 rows)
```

La fonction `generate_series` a été modifiée :

```
v12 $ \ef generate_series(bigint,bigint)

CREATE OR REPLACE FUNCTION pg_catalog.generate_series(bigint, bigint)
 RETURNS SETOF bigint
 LANGUAGE internal
 IMMUTABLE PARALLEL SAFE STRICT SUPPORT generate_series_int8_support
AS $function$generate_series_int8$function$
```

La fonction d'appui est appelée par l'optimiseur, pour retourner une estimation du nombre de lignes qui est calculée en rapport avec les 2 entiers donnés en paramètres.

2.4.2 TP REINDEX CONCURRENTLY

Les tests sont à effectuer sur une instance de PostgreSQL 12 avec le paramétrage par défaut.

- Créer 10 000 000 de lignes. La commande ci-dessous ajoute 10 millions d'enregistrements dans la table pgbench_accounts.

```
$ pgbench -i -s 100
```

- Réindexer la table sans `concurrently`.

Exemple de résultat (sans concurrently) :

```
$ time -f%E reindexdb -i pgbench_accounts_pkey
0:06.83
$ time -f%E reindexdb -i pgbench_accounts_pkey
0:06.85
```

```
$ time -f%E reindexdb -i pgbench_accounts_pkey
0:06.83
```

- Réindexer la table avec `concurrently`.

Exemple de résultat (avec concurrently) :

```
$ time -f%E reindexdb --concurrently -i pgbench_accounts_pkey
0:09.58
$ time -f%E reindexdb --concurrently -i pgbench_accounts_pkey
0:09.40
$ time -f%E reindexdb --concurrently -i pgbench_accounts_pkey
0:09.34
```

La réindexation classique est plus rapide car elle effectue moins d'opérations.

- Lancer un test avec `pgbench` de 20 secondes en lecture et déclencher une réindexation 5 secondes après le début du bench.

```
$ (sleep 5; time -f%E reindexdb -i pgbench_accounts_pkey ) & pgbench -c1 -T20 -S -P1 -r
```

Exemple de résultat :

```
starting vacuum...end.
progress: 1.0 s, 6375.9 tps, lat 0.156 ms stddev 0.029
progress: 2.0 s, 6549.7 tps, lat 0.152 ms stddev 0.016
progress: 3.0 s, 6586.0 tps, lat 0.151 ms stddev 0.019
progress: 4.0 s, 6902.0 tps, lat 0.145 ms stddev 0.020
progress: 5.0 s, 6603.9 tps, lat 0.148 ms stddev 0.017
progress: 6.0 s, 0.0 tps, lat 0.000 ms stddev 0.000    <- REINDEX en cours
progress: 7.0 s, 0.0 tps, lat 0.000 ms stddev 0.000    <- REINDEX en cours
....
0:06.70                                                <- temps de REINDEX
progress: 12.0 s, 2324.2 tps, lat 3.022 ms stddev 138.282
progress: 13.0 s, 6684.0 tps, lat 0.149 ms stddev 0.028
...
transaction type: <builtin: select only>
scaling factor: 100
query mode: simple
number of clients: 1
number of threads: 1
duration: 20 s
number of transactions actually processed: 89110
latency average = 0.224 ms
latency stddev = 22.327 ms
```

```
tps = 4455.484635 (including connections establishing)
tps = 4455.974731 (excluding connections establishing)
[1]+  Done       ( sleep 5; time -f%E reindexdb -i pgbench_accounts_pkey )
```

- Lancer un test avec pgbench de 20 secondes en lecture et déclencher une réindexation avec concurrently 5 secondes après le début du bench.

```
$ (sleep 5; time -f%E reindexdb --concurrently -i pgbench_accounts_pkey ) \
& pgbench -c1 -T20 -S -P1 -r
```

Exemple de résultat :

```
starting vacuum...end.
progress: 1.0 s, 6346.8 tps, lat 0.156 ms stddev 0.029
progress: 2.0 s, 6688.0 tps, lat 0.149 ms stddev 0.016
progress: 3.0 s, 6471.9 tps, lat 0.154 ms stddev 0.053
progress: 4.0 s, 6730.9 tps, lat 0.148 ms stddev 0.025
progress: 5.0 s, 6723.0 tps, lat 0.148 ms stddev 0.021
progress: 6.0 s, 4546.0 tps, lat 0.219 ms stddev 0.047  <- REINDEX concurrently
progress: 7.0 s, 4472.2 tps, lat 0.223 ms stddev 0.090  <- REINDEX concurrently
progress: 8.0 s, 4557.0 tps, lat 0.219 ms stddev 0.067  <- REINDEX concurrently
...
0:10.62                                                 <- temps de REINDEX
progress: 16.0 s, 6221.7 tps, lat 0.160 ms stddev 0.039
progress: 17.0 s, 6655.3 tps, lat 0.150 ms stddev 0.017
...
transaction type: <builtin: select only>
scaling factor: 100
query mode: simple
number of clients: 1
number of threads: 1
duration: 20 s
number of transactions actually processed: 120522
latency average = 0.165 ms
latency stddev = 0.150 ms
tps = 6026.060490 (including connections establishing)
tps = 6027.885412 (excluding connections establishing)
[1]+  Done    ( sleep 5; time -f%E reindexdb --concurrently -i pgbench_accounts_pkey )
```

- Lancer un test avec pgbench de 20 secondes en écriture et déclencher une réindexation 5 secondes après le début du bench.

```
$ (sleep 5; time -f%E reindexdb -i pgbench_accounts_pkey ) & pgbench -c1 -T20 -P1 -r
```

Exemple de résultat :

```
starting vacuum...end.
progress: 1.0 s, 146.0 tps, lat 6.769 ms stddev 1.448
progress: 2.0 s, 155.0 tps, lat 6.468 ms stddev 1.254
progress: 3.0 s, 129.0 tps, lat 7.773 ms stddev 1.439
progress: 4.0 s, 129.0 tps, lat 7.704 ms stddev 1.415
progress: 5.0 s, 156.0 tps, lat 6.310 ms stddev 1.235
progress: 6.0 s, 0.0 tps, lat 0.000 ms stddev 0.000 <- REINDEX en cours
progress: 7.0 s, 0.0 tps, lat 0.000 ms stddev 0.000
....
0:07.70
progress: 13.0 s, 48.0 tps, lat 167.134 ms stddev 1094.662
progress: 14.0 s, 126.0 tps, lat 7.894 ms stddev 1.741
progress: 15.0 s, 143.0 tps, lat 7.015 ms stddev 1.629
...
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 100
query mode: simple
number of clients: 1
number of threads: 1
duration: 20 s
number of transactions actually processed: 1746
latency average = 11.454 ms
latency stddev = 183.386 ms
tps = 87.274426 (including connections establishing)
tps = 87.294817 (excluding connections establishing)
statement latencies in milliseconds:
           0.004  \set aid random(1, 100000 * :scale)
           0.001  \set bid random(1, 1 * :scale)
           0.001  \set tid random(1, 10 * :scale)
           0.001  \set delta random(-5000, 5000)
           0.145  BEGIN;
           4.907  UPDATE pgbench_accounts SET abalance = abalance + :delta
                  WHERE aid = :aid;
           0.320  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
           0.419  UPDATE pgbench_tellers SET tbalance = tbalance + :delta
                  WHERE tid = :tid;
           0.367  UPDATE pgbench_branches SET bbalance = bbalance + :delta
                  WHERE bid = :bid;
```

```
          0.269  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
                 VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
          5.020  END;
[1]+  Done       () sleep 5; time -f%E reindexdb -i pgbench_accounts_pkey )
```

- Lancer un test avec pgbench de 20 secondes en écriture et déclencher une réindexation avec concurrently 5 secondes après le début du bench.

```
$ (sleep 5; time -f%E reindexdb --concurrently -i pgbench_accounts_pkey ) \
& pgbench -c1 -T20 -P1 -r
```

Exemple de résultat :

```
progress: 1.0 s, 155.0 tps, lat 6.412 ms stddev 1.510
progress: 2.0 s, 142.0 tps, lat 7.039 ms stddev 1.780
progress: 3.0 s, 126.0 tps, lat 7.955 ms stddev 1.673
progress: 4.0 s, 130.0 tps, lat 7.665 ms stddev 1.640
progress: 5.0 s, 146.0 tps, lat 6.870 ms stddev 1.580
progress: 6.0 s, 147.0 tps, lat 6.762 ms stddev 1.278
progress: 7.0 s, 144.0 tps, lat 6.934 ms stddev 1.284
...

0:11.89
progress: 17.0 s, 157.0 tps, lat 6.321 ms stddev 1.546
progress: 18.0 s, 139.0 tps, lat 7.188 ms stddev 1.555
progress: 19.0 s, 145.0 tps, lat 6.944 ms stddev 3.168
progress: 20.0 s, 147.0 tps, lat 6.768 ms stddev 1.444
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 100
query mode: simple
number of clients: 1
number of threads: 1
duration: 20 s
number of transactions actually processed: 2740
latency average = 7.297 ms
latency stddev = 6.863 ms
tps = 136.994259 (including connections establishing)
tps = 137.018048 (excluding connections establishing)
[1]+  Done ( sleep 5; time -f%E reindexdb --concurrently -i pgbench_accounts_pkey)
```

3 LICENSE

GNU Free Documentation License

Version 1.2, November 2002

```
Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document

that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the

body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque

copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

`A. Use in the Title Page (and on the covers, if any) a title distinct from`

that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed

in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section

unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”. 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
```

Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with…Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

NOTES

NOTES

NOS AUTRES PUBLICATIONS

FORMATIONS

- **DBA1 : Administration PostgreSQL**
  https://dali.bo/dba1
- **DBA2 : Administration PostgreSQL avancé**
  https://dali.bo/dba2
- **DBA3 : Sauvegardes et réplication avec PostgreSQL**
  https://dali.bo/dba3
- **DEVPG : Développer avec PostgreSQL**
  https://dali.bo/devpg
- **DEVSQLPG : SQL pour PostgreSQL**
  https://dali.bo/devsqlpg
- **PERF1 : PostgreSQL Performances**
  https://dali.bo/perf1
- **PERF2 : Indexation et SQL avancé**
  https://dali.bo/perf2
- **MIGORPG : Migrer d'Oracle vers PostgreSQL**
  https://dali.bo/migorpg

LIVRES BLANCS

- **Migrer d'Oracle à PostgreSQL**
- **Industrialiser PostgreSQL**
- **Bonnes pratiques de modélisation avec PostgreSQL**
- **Bonnes pratiques de développement avec PostgreSQL**

TÉLÉCHARGEMENT GRATUIT

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open-source ou sous licence Creative Commons. Contactez-nous à l'adresse contact@dalibo.com pour plus d'information.

DALIBO, L'EXPERTISE POSTGRESQL

Depuis 2005, DALIBO met à la disposition de ses clients son savoir-faire dans le domaine des bases de données et propose des services de conseil, de formation et de support aux entreprises et aux institutionnels.

En parallèle de son activité commerciale, DALIBO contribue aux développements de la communauté PostgreSQL et participe activement à l'animation de la communauté francophone de PostgreSQL. La société est également à l'origine de nombreux outils libres de supervision, de migration, de sauvegarde et d'optimisation.

Le succès de PostgreSQL démontre que la transparence, l'ouverture et l'auto-gestion sont à la fois une source d'innovation et un gage de pérennité. DALIBO a intégré ces principes dans son ADN en optant pour le statut de SCOP : la société est contrôlée à 100 % par ses salariés, les décisions sont prises collectivement et les bénéfices sont partagés à parts égales.