

CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



CHARGEMENT DE MASSE



```
pgday=# \x
Expanded display is on.
pgday=# select * from anthony ;
-[ RECORD 1 ]-----
nom          | Anthony Nowocien
twitter      | @nthonynowocien
poste        | DBA PostgreSQL
employeur    | SG
interets     | {photo,échecs,théâtre,...}
slides      | frama.link/chargez
```

Time: 0,592 ms

```
pgday=# table pourquoi_cette_presentation ;
```

```
id |          raison
```

```
----+-----
```

```
1 | Importance en entreprise
```

```
2 | Peu abordé dans les conférences
```

```
3 | REX suite à migrations
```

```
(3 rows)
```

```
Time: 0,425 ms
```

INSERT

COPY

OUTILS TIERS

INSERT

INSERT synopsis

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]  
    [ OVERRIDING { SYSTEM | USER } VALUE ]  
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }  
    [ ON CONFLICT [ conflict_target ] conflict_action ]  
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

where `conflict_target` can be one of:

```
    ( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opclass ] [, ...] )  
[ WHERE index_predicate ]  
    ON CONSTRAINT constraint_name
```

and `conflict_action` is one of:

```
DO NOTHING  
DO UPDATE SET { column_name = { expression | DEFAULT } |  
                ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |  
                ( column_name [, ...] ) = ( sub-SELECT )  
            } [, ...]  
[ WHERE condition ]
```

INSERT “de base”

```
INSERT INTO version(numero, release_date, eol, comment)
VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');
```

INSERT “de base”

```
INSERT INTO version(numero, release_date, eol, comment)
VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');
```

```
INSERT INTO version(numero, release_date, eol, comment)
VALUES ('10', '2016-09-26', '2021-11-11', 'Still maintained :)');
```

[...]

```
INSERT INTO version(numero, release_date, eol, comment)
VALUES ('6.3', '1998-03-01', '2003-03-01', 'Consider upgrading');
```

INSERT “de base”

```
BEGIN;  
INSERT INTO version(numero, release_date, eol, comment)  
    VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');  
COMMIT;
```

```
BEGIN;  
INSERT INTO version()  
    VALUES ();  
COMMIT;
```

```
BEGIN;  
INSERT INTO version()  
    VALUES ();  
COMMIT;
```

...

INSERT “de base”

BEGIN;

```
INSERT INTO version(numero, release_date, eol, comment)
VALUES ('11', '2023-10-18', '2023-11-09', 'latest stable');
COMMIT;
```

Parsing

Affectation d'un id de transaction virtuel

BEGIN;

```
INSERT INTO version()
VALUES ();
```

COMMIT;

BEGIN;

```
INSERT INTO version()
VALUES ();
```

COMMIT;

...

INSERT “de base”

```
BEGIN;  
INSERT INTO version(numero, release_date, eol, comment)  
VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');  
COMMIT;
```

```
BEGIN;  
INSERT INTO version()  
VALUES ();  
COMMIT;
```

```
BEGIN;  
INSERT INTO version()  
VALUES ();  
COMMIT;
```

...

DML !


Affectation d'un id de transaction

Génération d'un plan d'exécution

Exécution du plan

INSERT “de base”

```
BEGIN;  
INSERT INTO version(numero, release_date, eol, comment)  
VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');  
COMMIT;
```



```
BEGIN;  
INSERT INTO version()  
VALUES ();  
COMMIT;
```

Enregistrement dans un bloc
Ecriture dans le commit log
[...zzz...]

Ecriture dans les journaux de transaction

```
BEGIN;  
INSERT INTO version()  
VALUES ();  
COMMIT;
```

...

“RBAR”

“Row By Agonizing Row”

Un INSERT pour une transaction

Existe aussi en pire si on ouvre une session

INSERT “pluri-valué”

```
INSERT INTO version VALUES
  ('11', '2018-10-18', '2023-11-09', 'latest stable !'),
  ('10', '2016-09-26', '2021-11-11', 'Still maintained :)'),
  [...],
  ('6.3', '1998-03-01', '2003-03-01', 'Consider upgrading');
```

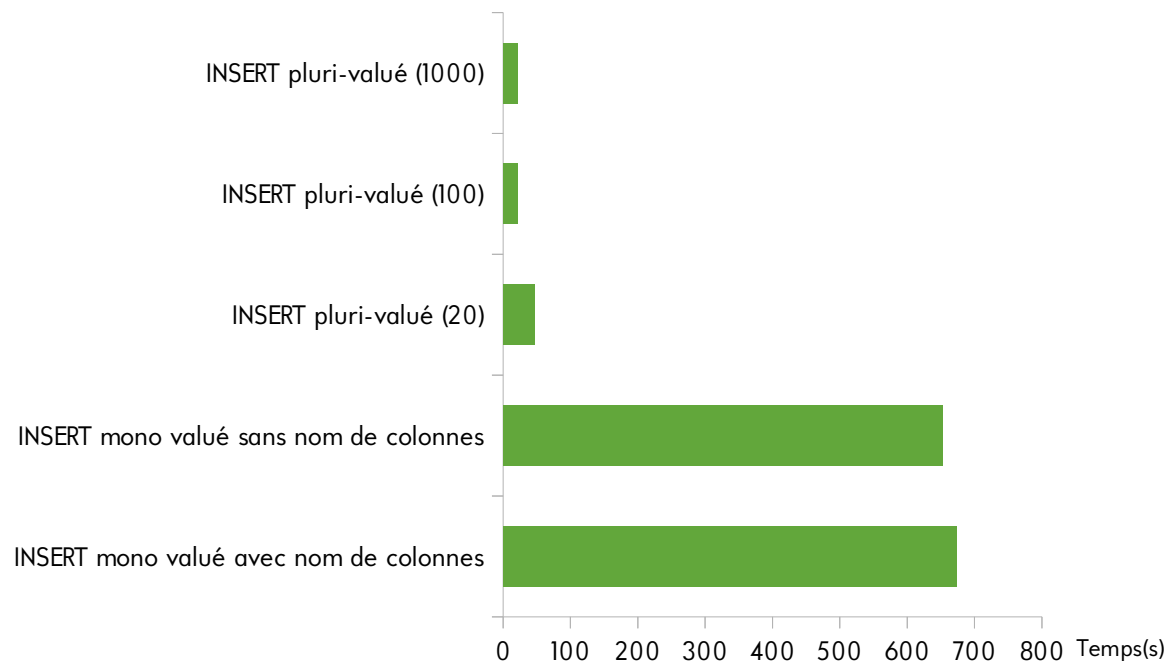
INSERT dans une transaction

```
BEGIN;  
INSERT INTO version VALUES ('11', '2018-10-18', '2023-11-09', 'latest stable');  
INSERT INTO version VALUES ('10', '2016-09-26', '2021-11-11', 'Still maintained');  
[...]  
INSERT INTO version VALUES ('6.3', '1998-03-01', '2003-03-01', 'Consider upgrading');  
COMMIT;
```

Type d'INSERT

1 000 000 lignes

Attention à trop de valeurs par insert



Partitionnement basé sur les triggers

```
CREATE TABLE measurement (  
    city_id int NOT NULL,  
    logdate date NOT NULL,  
    peaktemp int,  
    unitsales int  
);
```

```
CREATE TABLE measurement_y2006m02 (  
    CHECK (logdate >= DATE '2006-02-01' AND logdate < DATE '2006-03-01'))  
INHERITS (measurement);
```

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()  
RETURNS TRIGGER AS $$  
BEGIN  
    IF ( NEW.logdate >= DATE '2006-02-01' AND  
        NEW.logdate < DATE '2006-03-01' ) THEN  
        INSERT INTO measurement_y2006m02 VALUES (NEW.*);
```

Partitionnement basé sur les triggers

```
CREATE TABLE measurement (  
    city_id int NOT NULL,  
    logdate date NOT NULL,  
    peaktemp int,  
    unitsales int  
);
```

```
CREATE TABLE measurement_y2006m02 (  
    CHECK (logdate >= DATE '2006-02-01' AND logdate < DATE '2006-03-01'))  
INHERITS (measurement);
```

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()  
RETURNS TRIGGER AS $$  
BEGIN  
    IF ( NEW.logdate >= DATE '2006-02-01' AND  
        NEW.logdate < DATE '2006-03-01' ) THEN  
        INSERT INTO measurement_y2006m02 VALUES (NEW.*);
```



pg_partman !

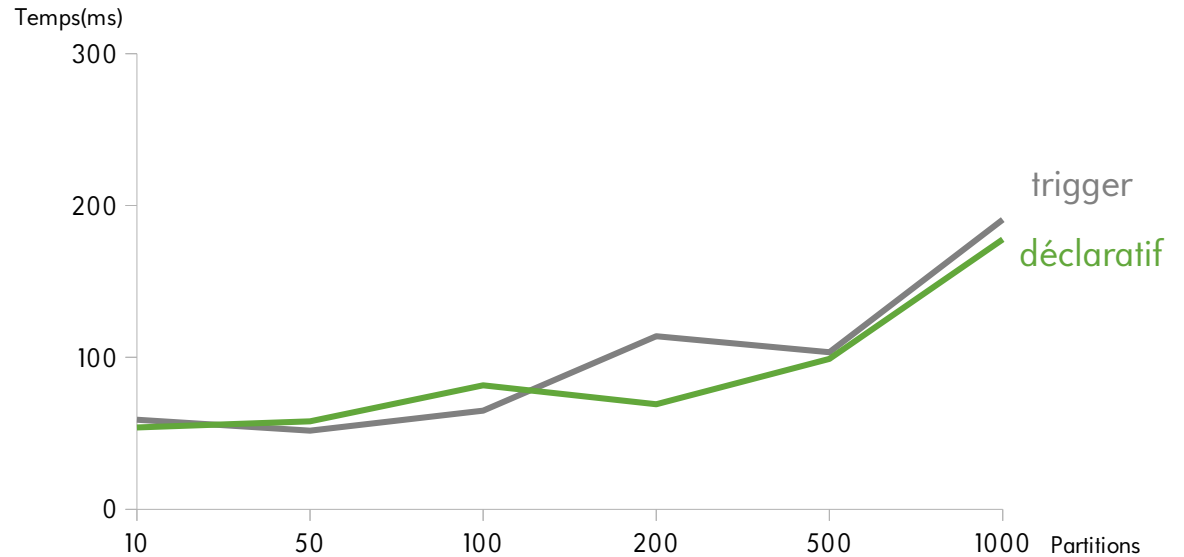
Partitionnement déclaratif

```
CREATE TABLE measurement (  
    city_id          int NOT NULL,  
    logdate          date NOT NULL,  
    peaktemp        int,  
    unitsales        int  
) PARTITION BY RANGE (logdate);
```

```
CREATE TABLE measurement_y2006m02 PARTITION OF measurement  
    FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
```

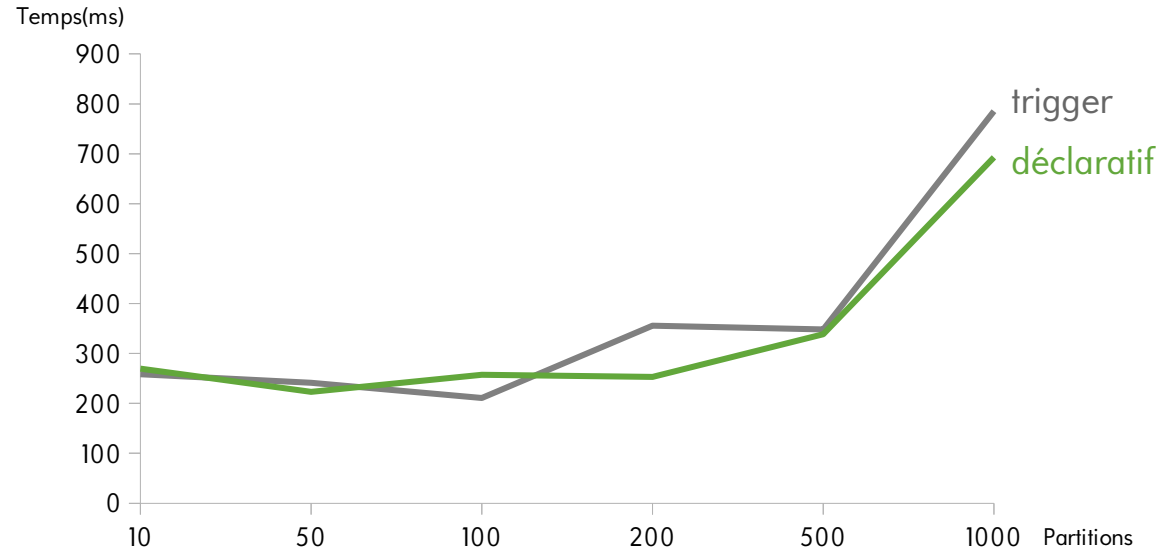

SELECT – table partitionnée

100 000 lignes



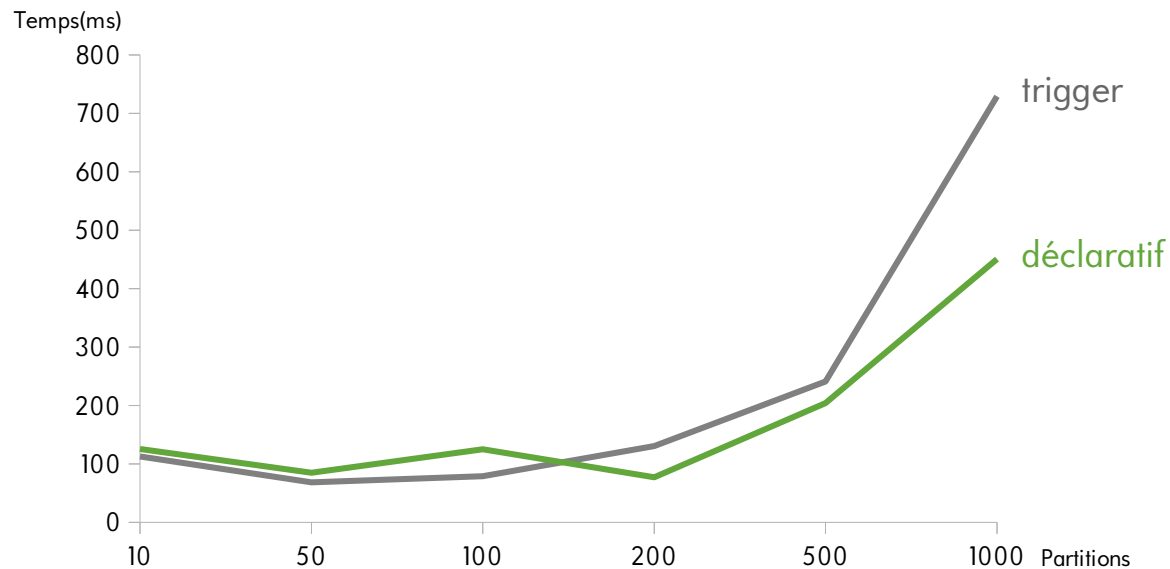
UPDATE – table partitionnée

100 000 lignes



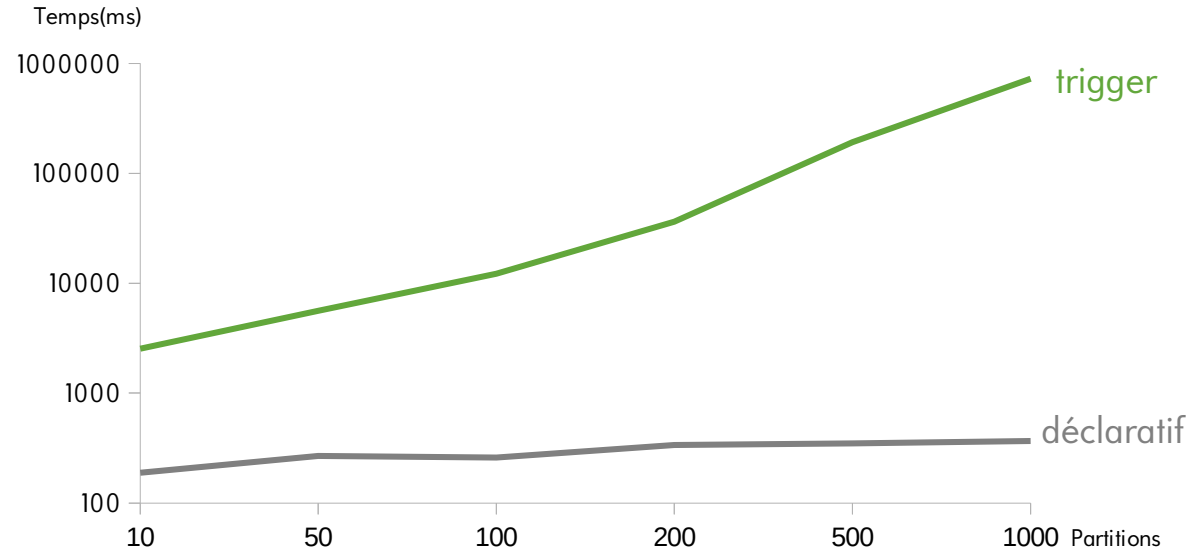
DELETE – table partitionnée

100 000 lignes



INSERT – table partitionnée

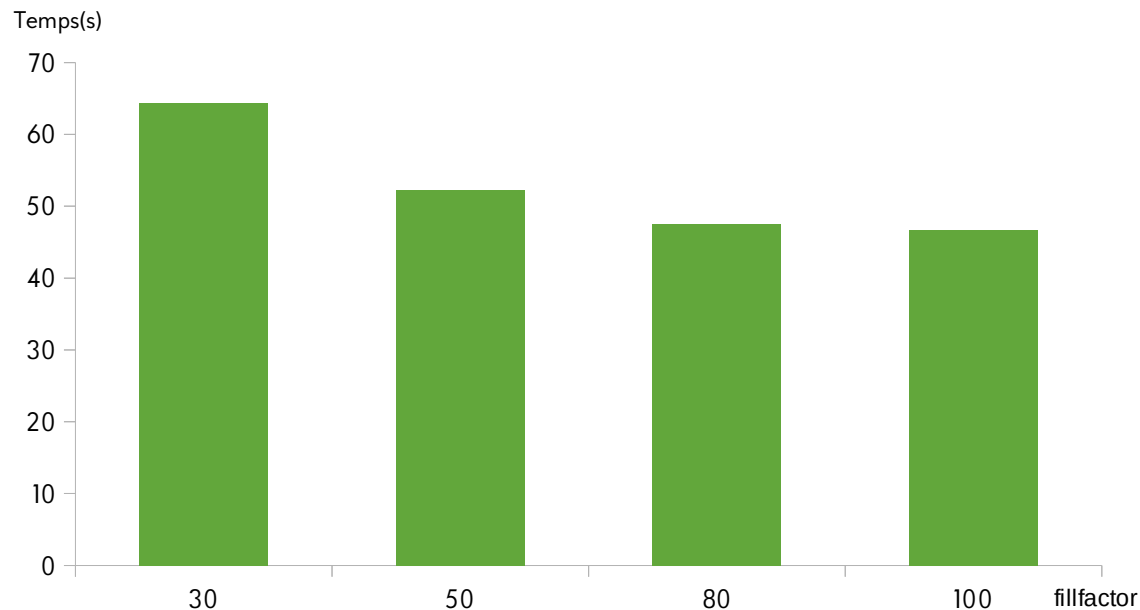
100 000 lignes



INSERT – fillfactor

1 000 000 lignes

```
ALTER TABLE t  
SET (fillfactor = 80);
```

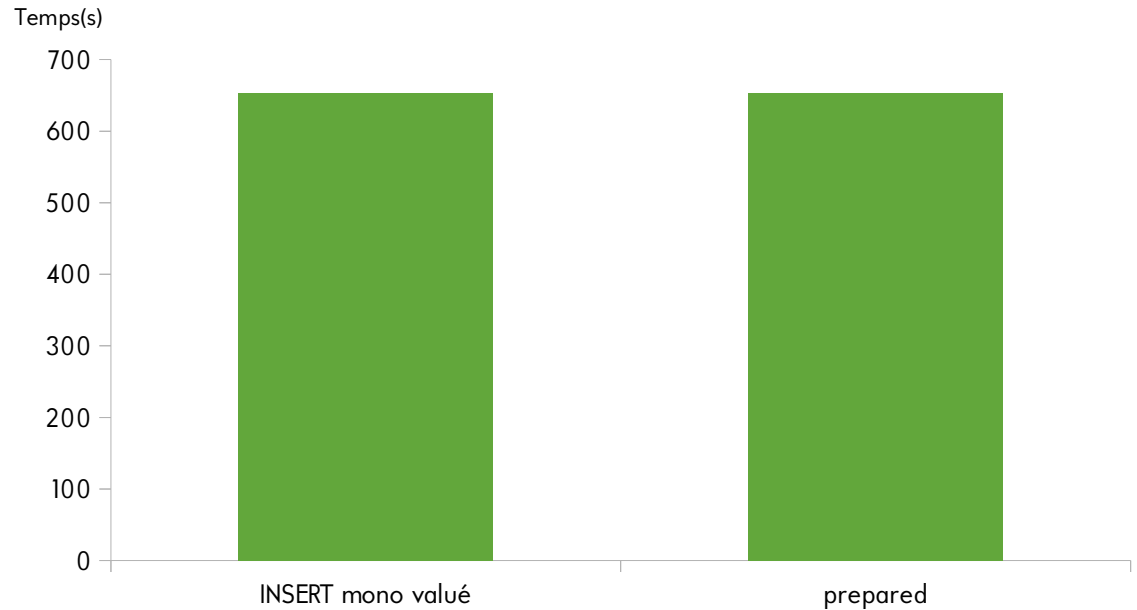


Voir aussi SQL-CREATETABLE-STORAGE-PARAMETERS:
toast_tuple_target, parallel_workers, autovacuum_enabled, ...

INSERT – prepare ?

```
PREPARE prepinsertt(int,text,text,int, te  
as INSERT INTO public.t  
values($1,$2,$3,$4,$5);  
EXECUTE prepinsertt (1, 'str1', 'str2', 1  
'str3');  
DEALLOCATE prepinsertt;
```

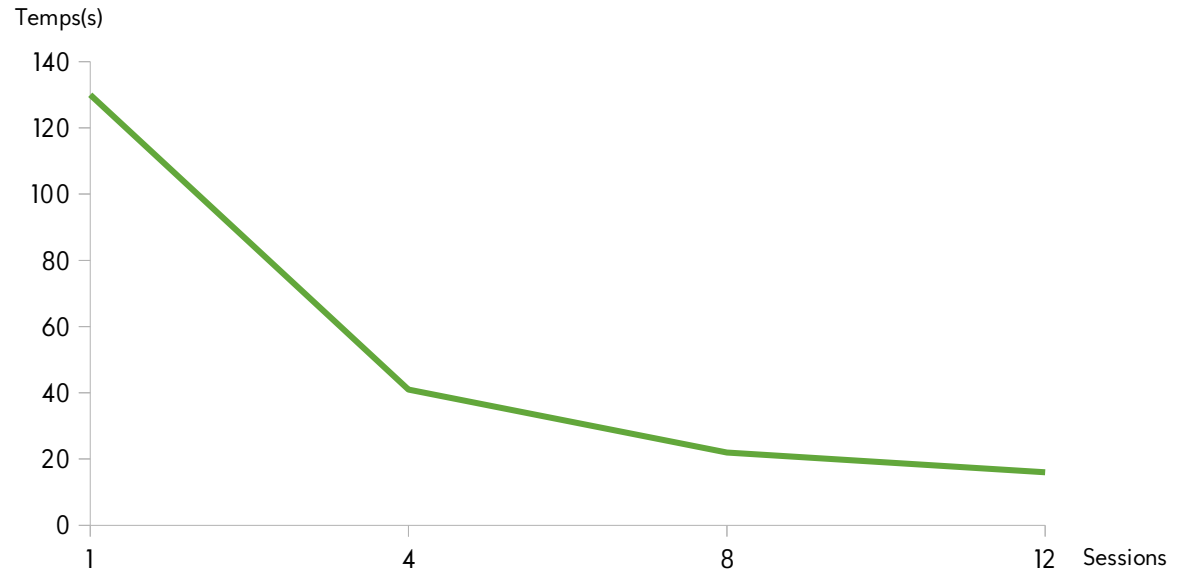
Prepared: moins de volatilité



INSERT – sessions multiples

ODBC

40 000 000 lignes



INSERT



PERFORMANCES



FONCTIONNALITES

.

UBIQUITE

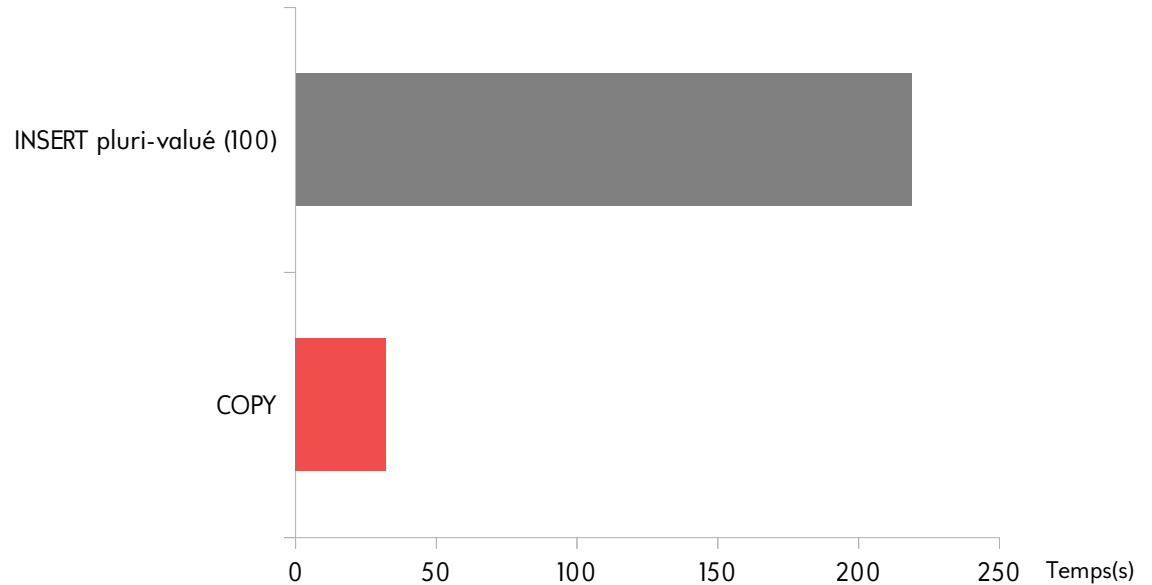
COPY

COPY ?

Oracle	SQL Server
sqlloader	bcp
DB2	MySQL
LOAD	LOAD DATA

Outil dédié de chargement

Déjà dans la v.1.0.8 (sortie en 1996) !



COPY - améliorations

1.0.8 : déjà présent

8.0: CSV avec DELIMITER, ESCAPE, QUOTE, ...

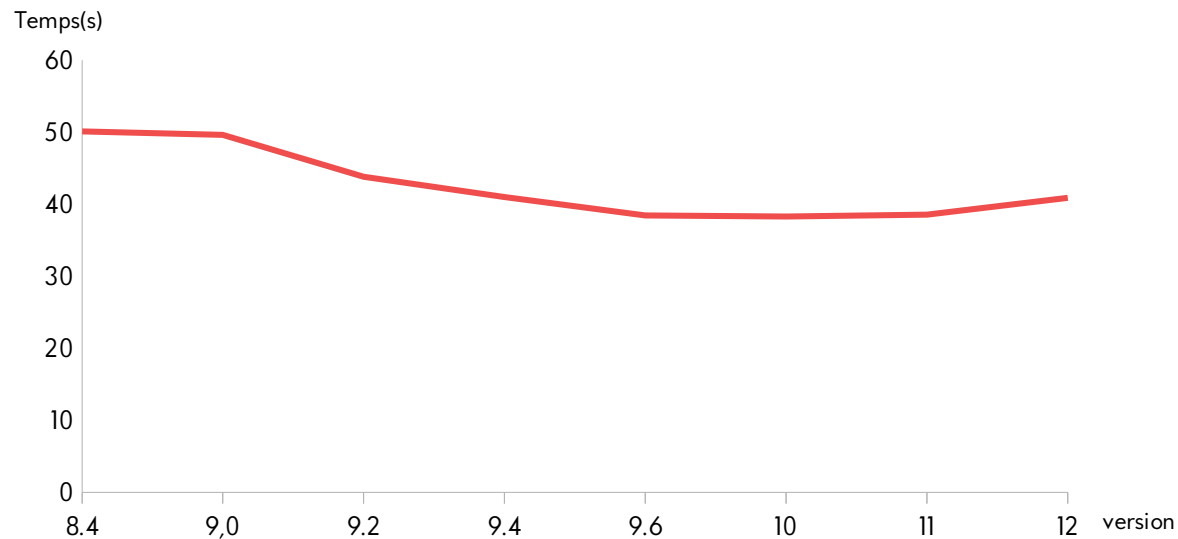
9.0: adaption de la syntaxe (BINARY, ...)

9.1: ENCODING

9.3: PROGRAM, FREEZE

9.4: FORCE NULL

12: COPY FROM WHERE



COPY - synopsis

```
COPY table_name [ ( column_name [, ...] ) ]  
  FROM { 'filename' | PROGRAM 'command' | STDIN }  
  [ [ WITH ] ( option [, ...] ) ]  
  [ WHERE condition ]
```

```
COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }  
  TO { 'filename' | PROGRAM 'command' | STDOUT }  
  [ [ WITH ] ( option [, ...] ) ]
```

where option can be one of:

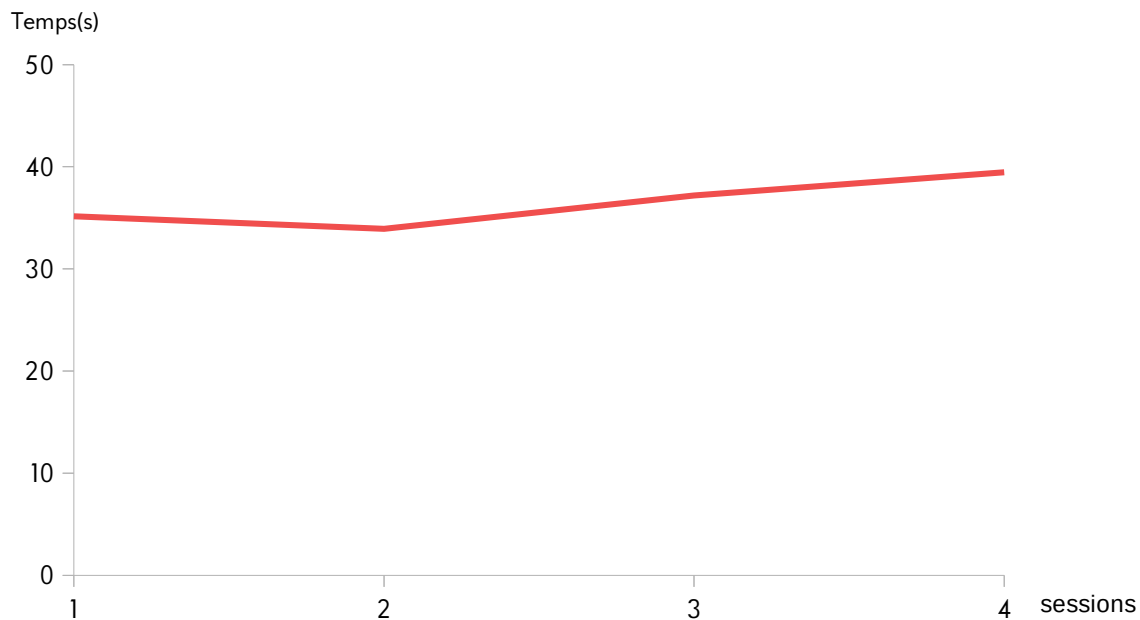
```
FORMAT format_name  
FREEZE [ boolean ]  
DELIMITER 'delimiter_character'  
NULL 'null_string'  
HEADER [ boolean ]  
QUOTE 'quote_character'  
ESCAPE 'escape_character'  
FORCE_QUOTE { ( column_name [, ...] ) | * }  
FORCE_NOT_NULL ( column_name [, ...] )  
FORCE_NULL ( column_name [, ...] )  
ENCODING 'encoding_name'
```

COPY plusieurs fichiers/sessions

1.3Go

10 000 000 lignes

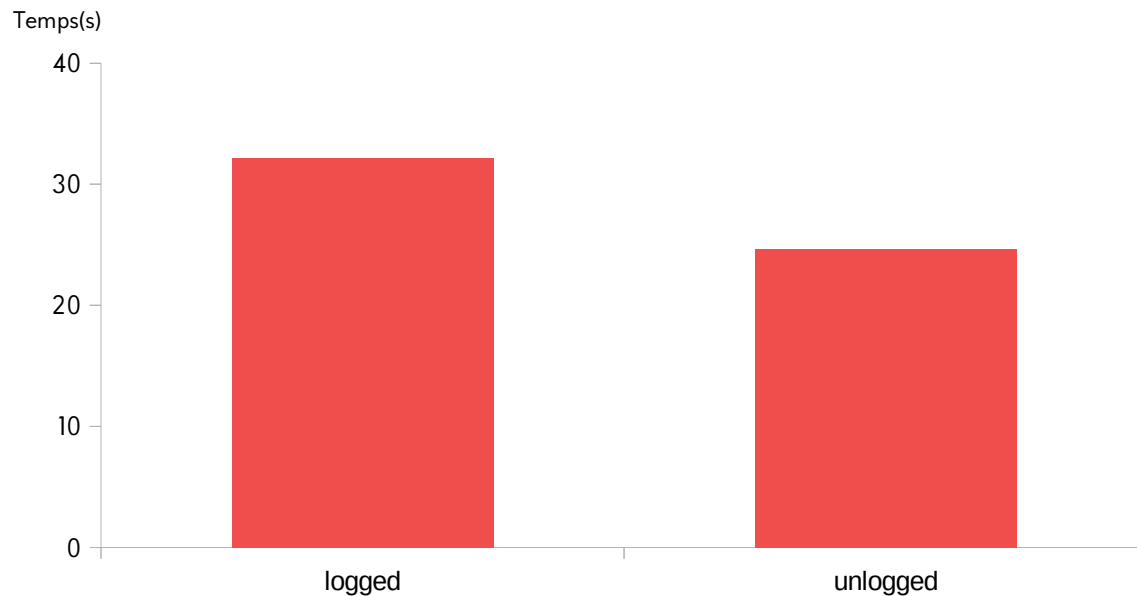
```
psql -c "\copy t from t4_0.out" bench &  
psql -c "\copy t from t4_1.out" bench &  
psql -c "\copy t from t4_2.out" bench &  
psql -c "\copy t from t4_3.out" bench &  
wait;
```



COPY – logguer ou pas

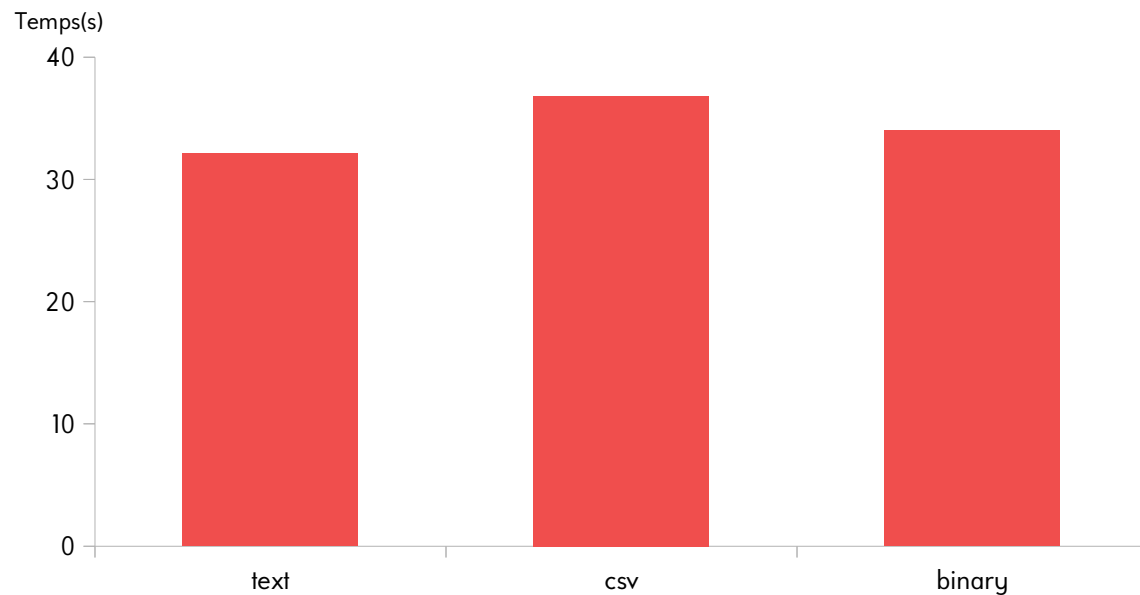
1.3Go
10 000 000 lignes

```
ALTER TABLE t SET LOGGED;  
ALTER TABLE t SET UNLOGGED;
```



COPY – format

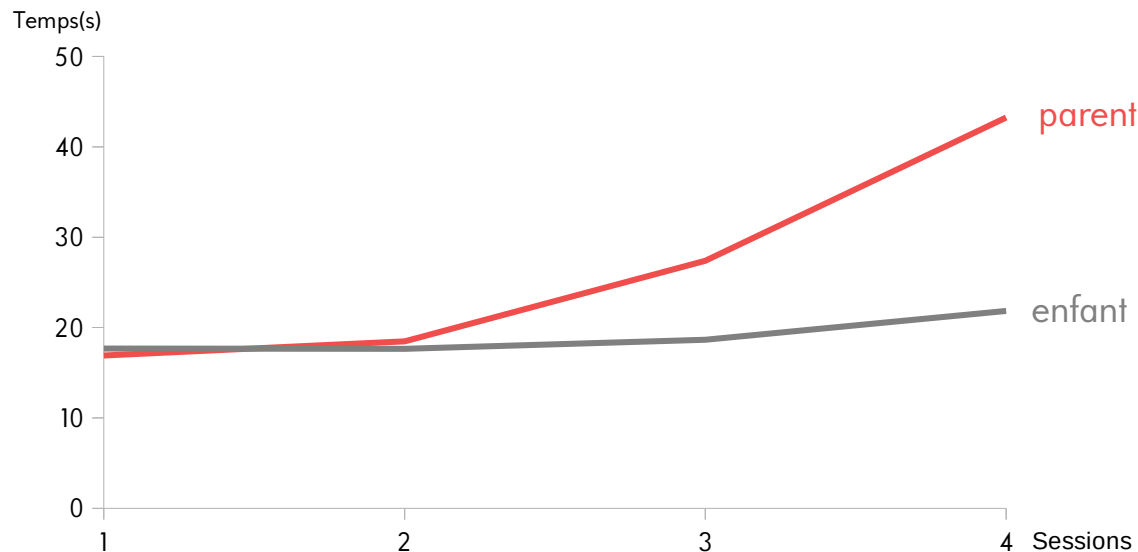
1.3Go
10 000 000 lignes



COPY – table partitionnée

nb de COPY en || sur table parente

1 000 000 sur table parente

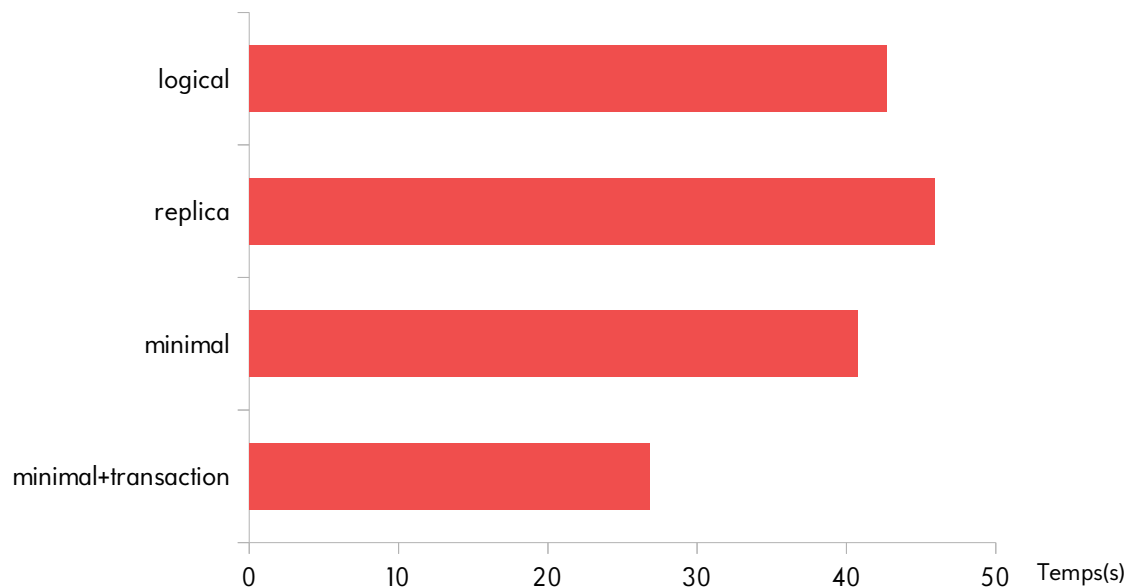
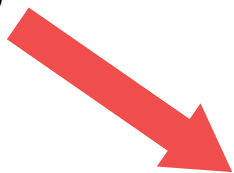


copy tpart from file where mod(a,4)=0

copy tpart from file where mod(a,4)=1

COPY – wal level

```
wal_level = minimal  
archive_mode = off  
max_wal_senders = 0
```



Voir <https://www.postgresql.org/docs/current/populate.html>

COPY FROM PROGRAM


Interdit au moins de 18 ans !

<https://blog.hagander.net/when-a-vulnerability-is-not-a-vulnerability-244/>

COPY sur fichier avec colonnes fixes :

```
COPY mytable FROM PROGRAM 'sed \"s/./&;/39;s/./&;/32;s/./&;/27;s/./&;/5\" /  
tmp/myfile.dat' DELIMITER ';' CSV
```

COPY = ETL ?



“I do not think this is a good idea.
We have resisted attempts to add ETL-like features to COPY on
the grounds that it would add complexity and cost performance, and that’s not
what COPY is for. This seems to fall squarely in the domain of
**something you should be doing
with another tool.**”

Regards,
Tom Lane

COPY - TODO

PostgreSQL 12

- greatly reduce memory consumption (andreas freund, tomas vondra, tom lane)
- copy from WHERE condition: <https://commitfest.postgresql.org/21/1824/>
- speed-up COPY FROM buffering

<https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=86b85044e823a304d2a265abc030254d39efe7df>

En cours <https://wiki.postgresql.org/wiki/ToDo#COPY>

- parallel COPY FROM: <https://commitfest.postgresql.org/14/1266/>
- COPY text format to output a header: <https://commitfest.postgresql.org/19/1629/>
- Allow COPY to report error lines and continue. This requires the use of a savepoint before each COPY line is processed, with ROLLBACK on COPY failure.

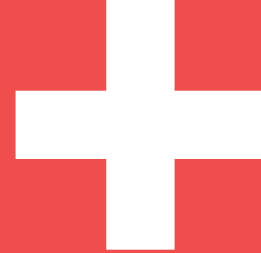
COPY



DRIVERS

.

TOUT OU RIEN



PERFORMANCES

.

INCLUS

Outils tiers

pg_loader

pgloader

Principes:

- Migration d'un autre SGBD

```
pgloader mysql://user@localhost/sakila postgresql:///pagila
```

- Transformer la donnée;
- Wrapper au dessus de COPY !



pgloader

```
$ pgloader --type csv t.csv postgres:///pgbench?tablename=t  
[...]
```

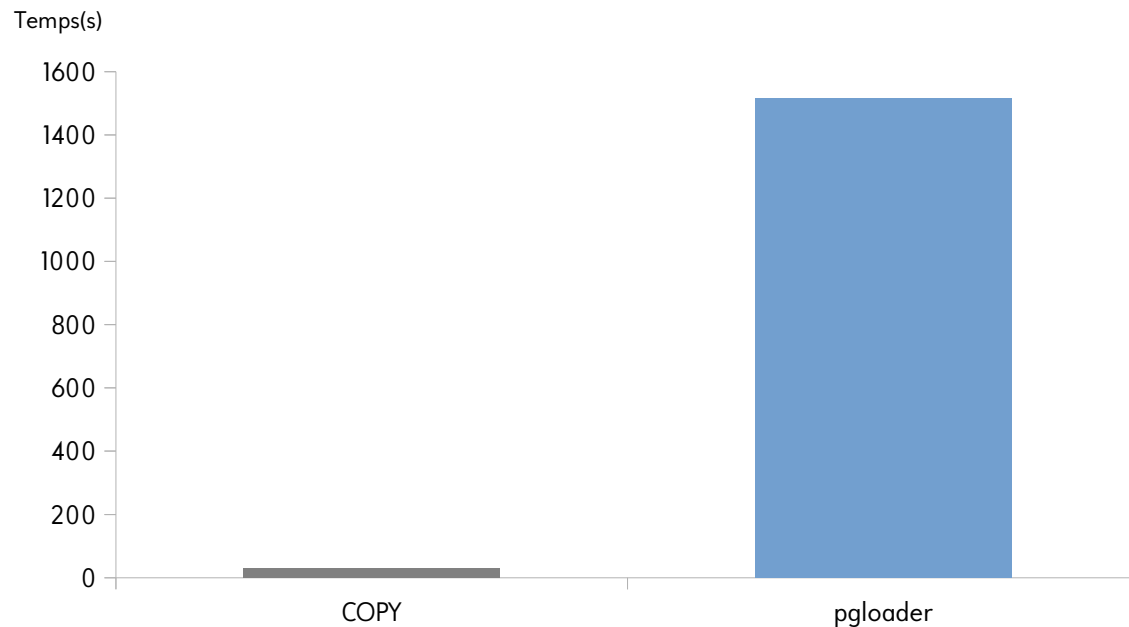
table name	errors	rows	bytes	total time
-----	-----	-----	-----	-----
fetch	0	0		0.010s
-----	-----	-----	-----	-----
"public"."t"	0	1000000	102.9 MB	2m36.277s
-----	-----	-----	-----	-----
Files Processed	0	1		0.026s
COPY Threads Completion	0	2		2m36.285s
-----	-----	-----	-----	-----
Total import time	✓	1000000	102.9 M	2m36.311s

pgloader

CSV de 1.3Go
10 000 000 lignes

pgloader 3.6.1

Paramètres par défaut pour csv



pgloader



PERFORMANCES (vs COPY)



FONCTIONNALITES

- REPRISE SI ERREUR

- MAJ

pg_bulkload

pg_bulkload

Principes:

- Améliorer COPY;
- Fait pour les gros volumes;
- Permettre certaines transformations;



Passe outre les `shared_buffers` !

pg_bulkload

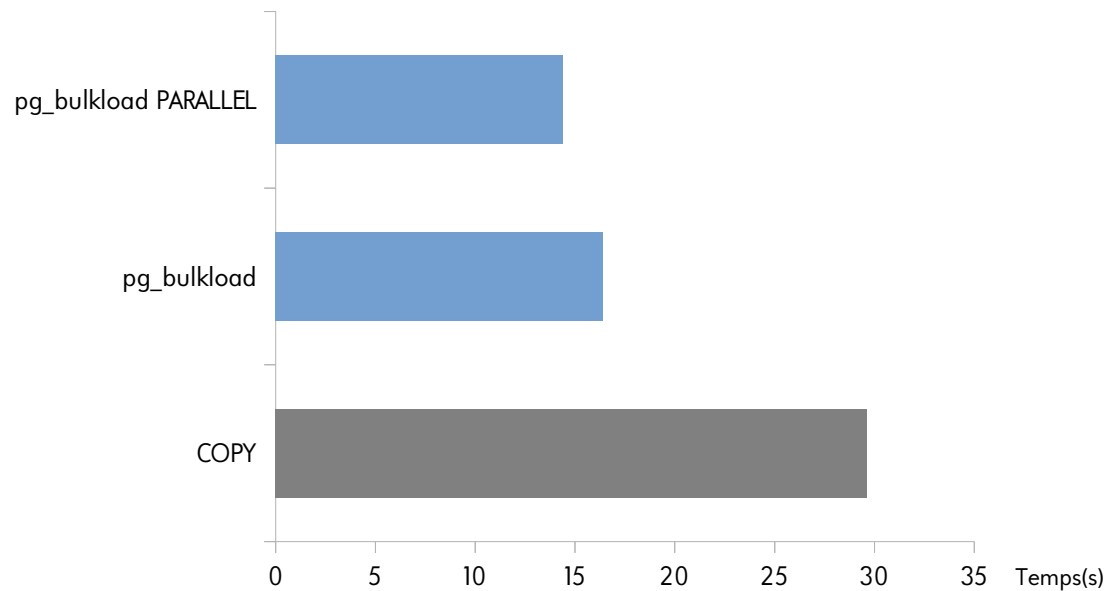
```
-bash-4.2$ cat pgbulkload_t.ctl
#
# sample_csv.ctl -- Control file to load CSV input data
#
# Copyright (c) 2007-2019, NIPPON TELEGRAPH AND TELEPHONE CORPORATION
#
OUTPUT = t                               # [<schema_name>.]table_name
INPUT = /tmp/t.csv # Input data location (absolute path)
TYPE = CSV                               # Input file type
QUOTE = "\""                             # Quoting character
ESCAPE = \                               # Escape character for Quoting
DELIMITER = ","                          # Delimiter
WRITER = PARALLEL                        # DIRECT | BUFFERED | BINARY | PARALLEL
```

pg_bulkload

```
-bash-4.2$ /usr/pgsql-10/bin/pg_bulkload /tmp/pgbulkload_t.ct1  
NOTICE: BULK LOAD START  
NOTICE: BULK LOAD END  
0 Rows skipped.  
10000000 Rows successfully loaded.  
0 Rows not loaded due to parse errors.  
0 Rows not loaded due to duplicate errors.  
0 Rows replaced with new rows.
```

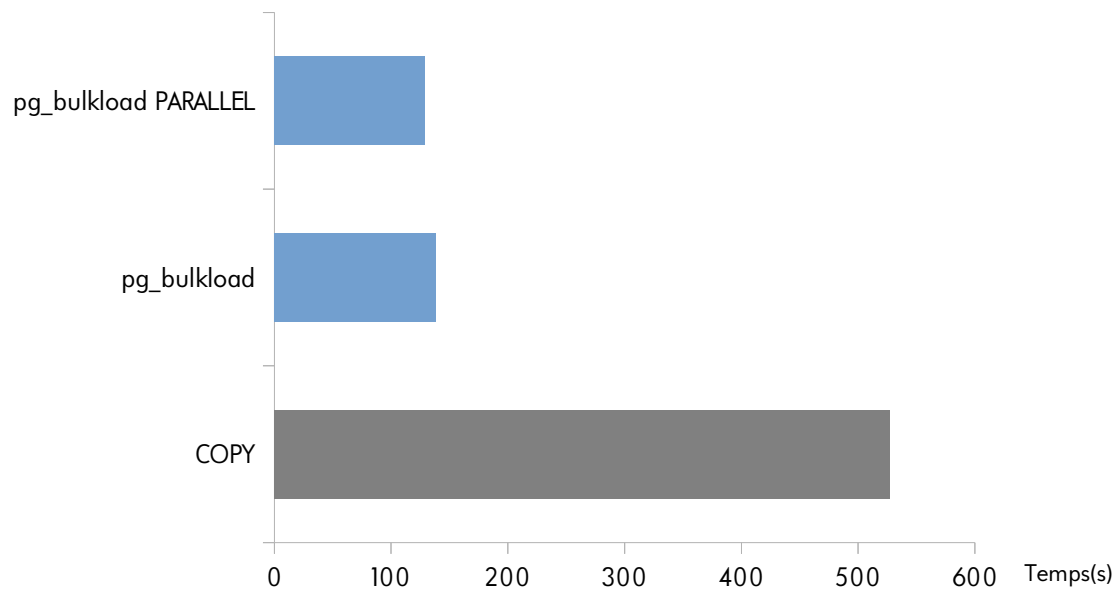

pg_bulkload – sans index

1.3Go
10 000 000 lignes



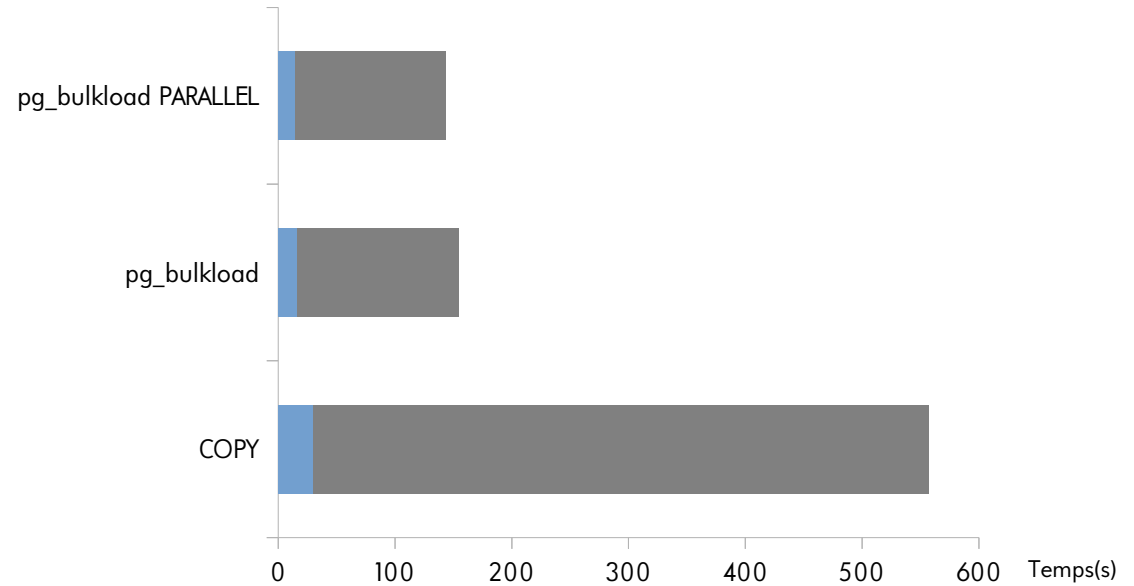
pg_bulkload – avec index

1.3Go
10 000 000 lignes



pg_bulkload – avec/sans index

1.3Go
10 000 000 lignes



pg_bulkload



FK

•

MAJ

•

DOCUMENTATION



PERFORMANCES

pg_discard

pg_discard

Principe:

- Dropper les contraintes
- Faire le chargement
- Recréer les contraintes
 - En cas d'erreur, déplacer les lignes dans une table *__rejected*.

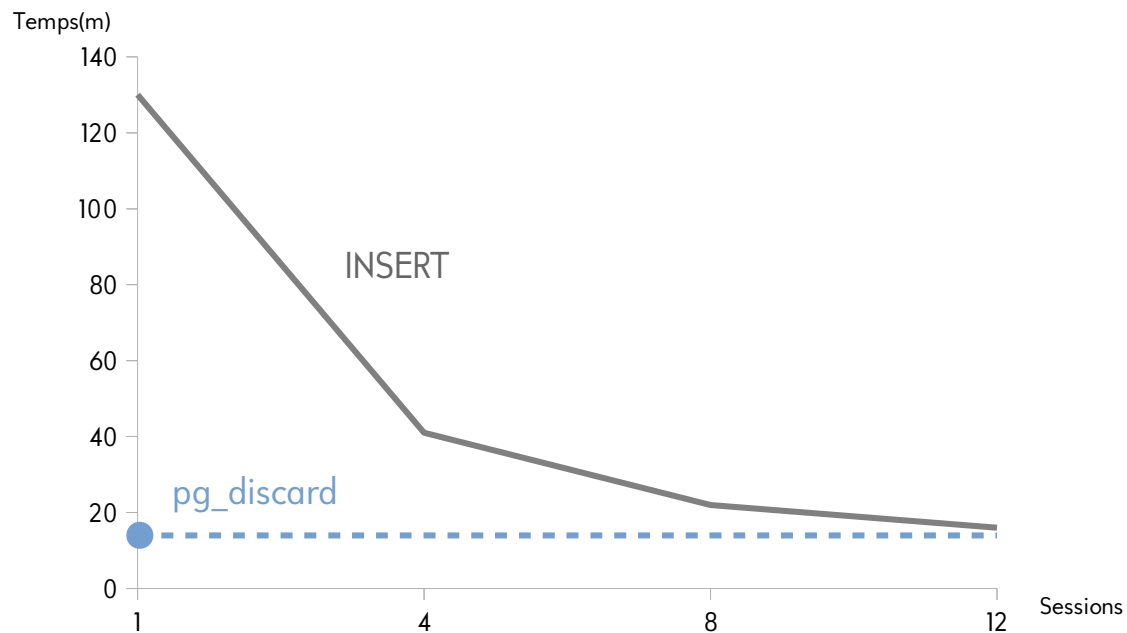
```
-bash-4.2$ ./pg_discard postgres:///mydb mytable drop  
[...]  
-bash-4.2$ BULK LOADING OPERATION  
[...]  
-bash-4.2$ ./pg_discard postgres:///mydb mytable add
```

pg_discard

INSERT sur plusieurs sessions

VS

pg_discard



pg_discard



TRES SPECIFIQUE

•

ALPHA



PERFORMANCES

Performances

ENABLE_ENTERPRISE_TURBO_BOOST = ON

Performances

~~ENABLE_ENTERPRISE_TURBO_BOOST = ON~~

Performances

RECREER INDEX
•
RECREER CONTRAINTES
•
RECREER TRIGGERS

MAX_WAL_SIZE
•
MAINTENANCE_WORK_MEM
•
SYNCHRONOUS_COMMIT

STATISTIQUES

Et ensuite?

Conclusion

Bien comprendre les besoins => bien choisir son outil

Paramétrez votre connecteur

Bench it !

Des manques ?

REX

Sponsoriser / développer des fonctionnalités manquantes ?

Vos questions