The evolution of declarative partitioning from PostgreSQL 10 to 14

About me

Julia Gugel

Consultant

+41 78 320 43 07

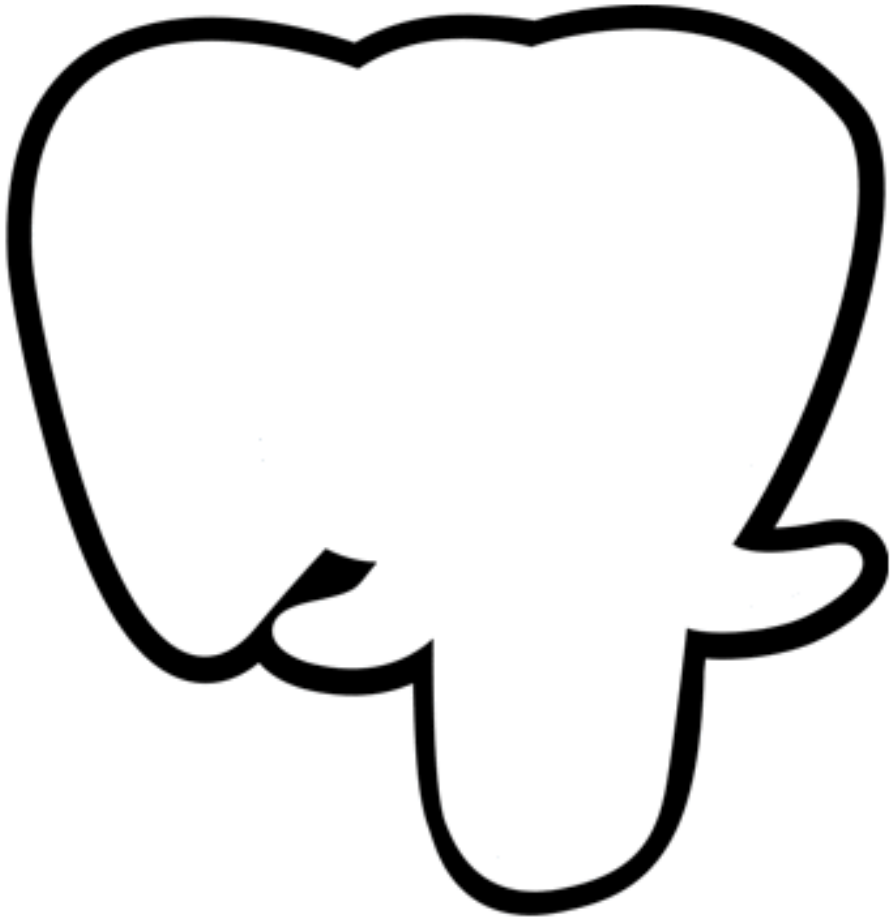
julia.gugel[at]dbi-services.com

@gugeljuli

Julia Gugel

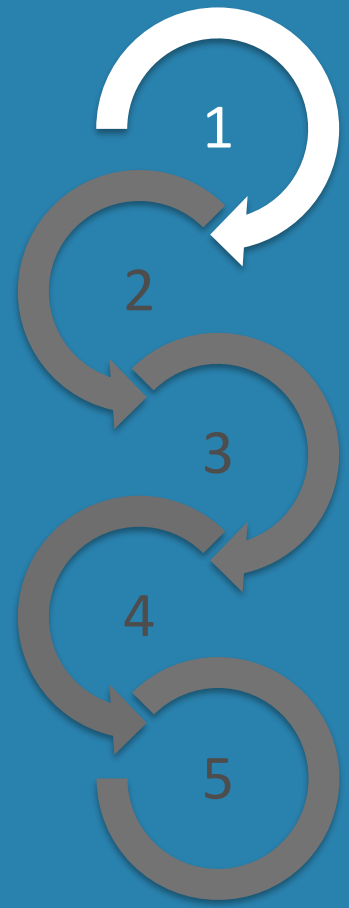
Agenda

1. Partitioning
2. Before declarative partitioning
3. Declarative partitioning
4. Demo
5. Conclusion



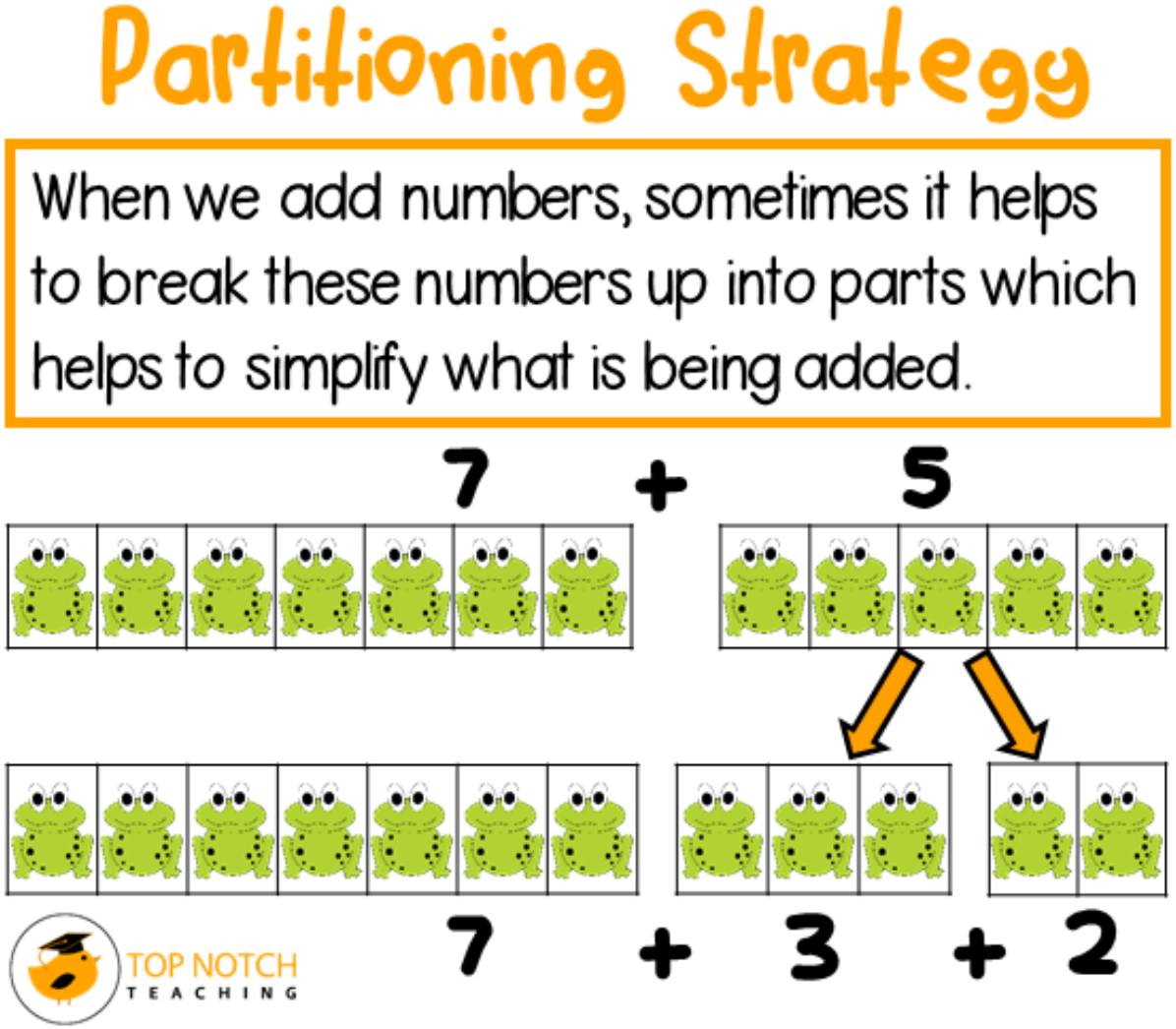
Partitioning

- What is partitioning?
- How PostgreSQL defines
- Why should I use partitioning?



Partitioning
What is partitioning?

What schools teach



Partitioning refers to splitting what is logically one large table into smaller physical pieces.

Partitioning
Why should I use partitioning?

Query performance
> Huge tables = huge indexes
> Whole data will be scanned
> Partitioning: smaller data chunks

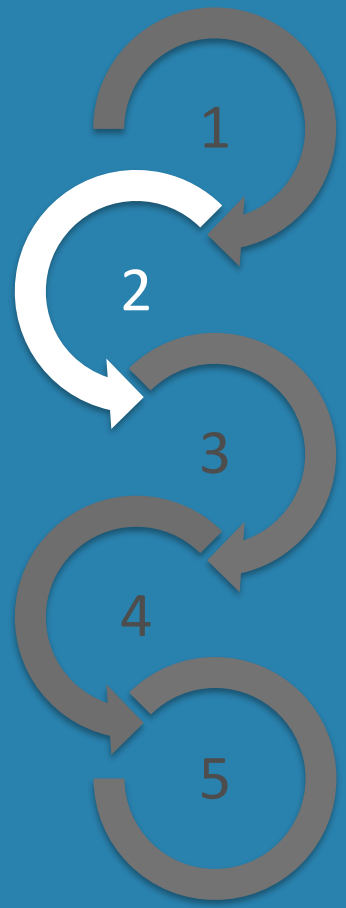
Deletion
> Delete old data might take ages
> Partitioning: Delete partitions

Costs
> Move seldom-used data to slower and cheaper storage

BUT
> Not every table will benefit from partitioning

Before declarative partitioning

- Inheritance
- Problems



Before declarative partitioning
Inheritance

Available since Version 8.1

PostgreSQLs 1st possibility to create partitions

> Range partitioning

> List partitioning

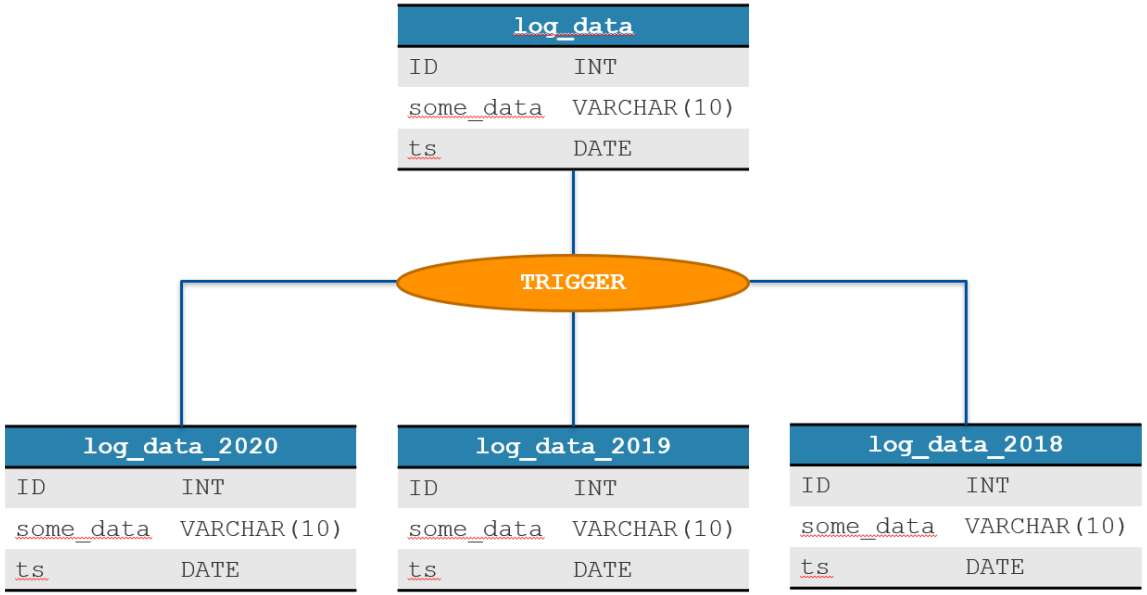
Implementation

> Create a parent table

> Create x child (partitioned) tables

> Create a check constraint on each child table

> Create a trigger so PostgreSQL knows where to route data



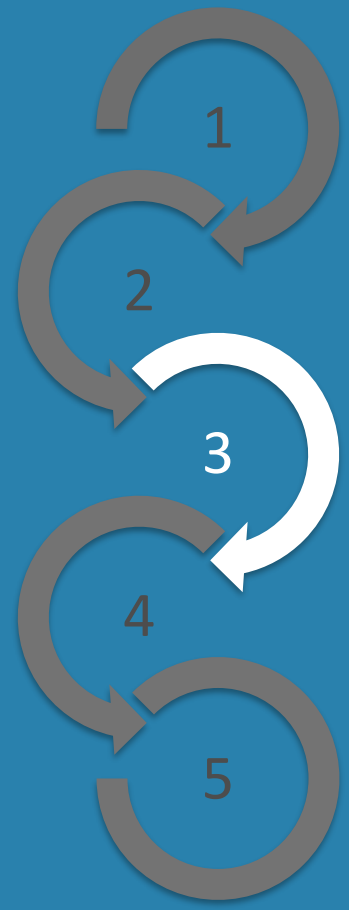
Before declarative partitioning
Problems

Missing
> Default partition
> Hash partitioning
> Create partitioned indexes
> No foreign keys on partitioned tables
> Automatic partition for incoming data

Problems
> Check constraints are not checked automatically to be exclusive
> VACUUM and ANALYZE need to run on each partition
> High administrative effort – rebuild function whenever the table changes
> Errors?

Declarative Partitioning

- What is declarative partitioning?
- Version 10
- Version 11
- Version 12
- Version 13



Declarative Partitioning
What is declarative partitioning?

PostgreSQL specific for:

> How to divide a table into pieces

Inserts into partitioned table routed to partitions

> based on the value of the partition key

Declarative Partitioning
Version 10

Now possible
> Specification for partitioning method and partition key
  > For each partition assigned
> Subpartitions
> Range and list partitioning

Not possible
> Turn regular table into a partitioned table or vice versa

But
> Add a regular table as a partition of a partitioned table

Declarative Partitioning
Version 10

Standard syntax

```
$ [ PARTITION BY { RANGE | LIST } ( { column_name | ( expression ) } [ COLLATE collation ]
[ opclass ] [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
```

Declarative Partitioning
Version 11

Hash Partitioning

Default partition

Indexes and constraints

> Indexes automatically created on partitioned tables

> Foreign keys automatically on partitioned tables

Partition Pruning

> Query optimization

> Prove if partition needs to be scanned

> Exclusion from query plan

```
$ SET enable_partition_pruning=on;
```

Declarative Partitioning
Version 11

DML improvements

> UPDATE statements that change a partition key column cause affected rows to move to the other partition

Triggers

> FOR EACH ROW triggers on partitioned tables

> Trigger on a partitioned table automatically creates triggers on all existing and future partitions

Declarative Partitioning
Version 11

Extended functionalities

> Allow faster partition elimination
> Allow matching partitions to be joined directly
> Allow aggregate functions on partitioned tables to be evaluated separately

Declarative Partitioning
Version 12

New partition introspection functions

> pg_partition_root()
> pg_partition_ancestors()
> pg_partition_tree()

psql improvements

> \dP to list partitioned tables and indexes

Declarative Partitioning
Version 12

Foreign keys to reference partitioned tables

Partition bounds can be any expression

> Expressions are evaluated at partitioned-table creation time

> Before it were only simple constants

Avoid sorting when partitions are already scanned

Declarative Partitioning
Version 13

Improvements
> Better performance for queries on partitioned tables
> Partition wise join happens more frequently
> Partition pruning happens more frequently
> Prevent dropping a tablespace referenced by partition relation

New
> Before ROW-LEVEL triggers
> Logical replication of partitioned tables via publications
> Logical replication into partitioned tables on subscribers
> Whole-row variables used in partitioned expressions

Declarative Partitioning
Version 14

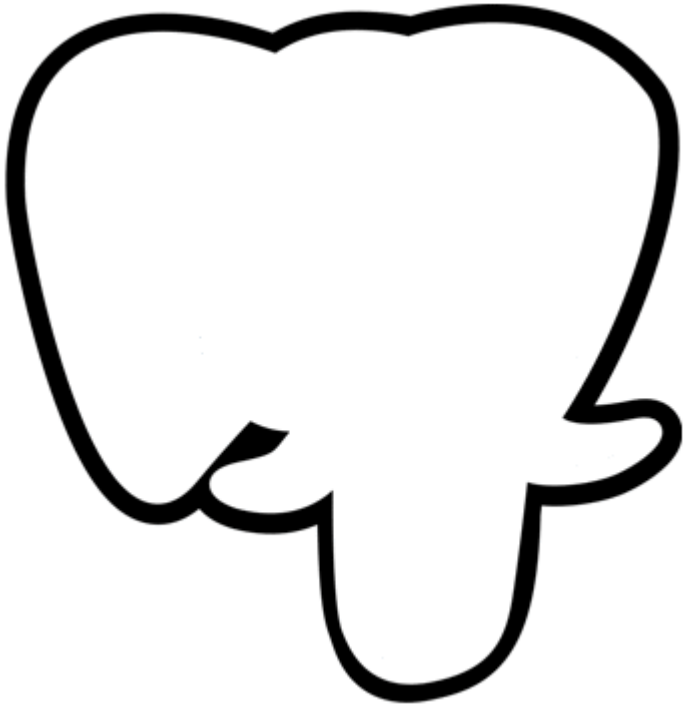
Detach partitions without blocking

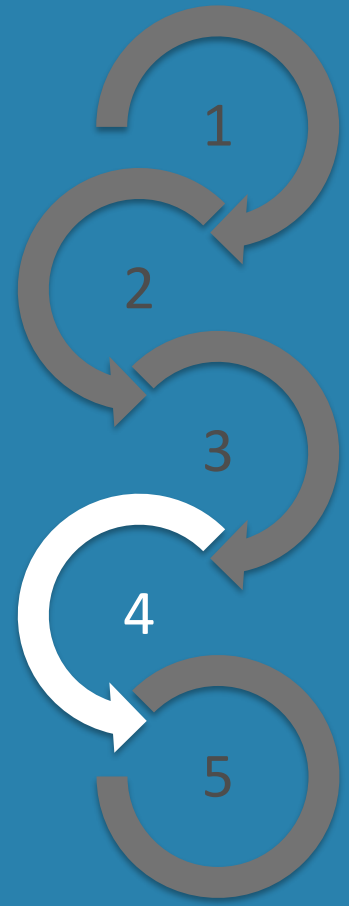
Improve performance of updates/deletes

> When only a few partitions are affected

```
$ ALTER TABLE...DETACH PARTITION...CONCURRENTLY;
```

Demo





Demo
PostgreSQL 9

Commands for the demo on PostgreSQL Version 9

```
/* Create a simple table */
create table log_data ( id int, some_data varchar(10), ts date );

/* Create three table that inherits from log_data */
create table log_data_2020() inherits ( log_data );
create table log_data_2019() inherits ( log_data );
create table log_data_2018() inherits ( log_data );

/* Describe the table structure */
\d+ log_data_2020
\d+ log_data_2019
\d+ log_data_2018
\d+ log_data
```

Commands for the demo on PostgreSQL Version 9

```
/* Create a function to split the data */

CREATE OR REPLACE FUNCTION log_data_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
  IF ( NEW.ts >= DATE '2019.01.01' AND
     NEW.ts < DATE '2020-01-01' ) THEN INSERT INTO log_data_2019 VALUES (NEW.*);
  ELSIF ( NEW.ts >= DATE '2020-01-01' AND
     NEW.ts < DATE '2021-01-01' ) THEN
        INSERT INTO log_data_2020 VALUES (NEW.*);
        ELSE
        RAISE EXCEPTION 'Date out of range!';
  END IF;
RETURN NULL;
END;
$$
 LANGUAGE plpgsql;
```

Commands for the demo on PostgreSQL Version 9

```
/* Create a trigger to split the data */
CREATE TRIGGER insert_log_data_trigger
   BEFORE INSERT ON log_data
FOR EACH ROW EXECUTE PROCEDURE log_data_insert_trigger();

/* Insert data into log_data */
insert into log_data values ( 1, 'aaaa', date('2019.03.03'));
insert into log_data values ( 2, 'aaaa', date('2020.03.03'));
insert into log_data values ( 2, 'aaaa', date('2021.03.03'));

/* See the data spreading */
select * from log_data;
select * from log_data_2019;
select * from log_data_2020;

explain analyze select * from log_data where ts = date ('2020.03.03');
```

Commands for the demo on PostgreSQL Version 10

```
/* Select the smallest and biggest date to know how to partition */
select min(date_of_stop), max(date_of_stop) from mv_traffic_violations;

/* Create a table with range partitioning on column date_of_stop */
create table traffic_violations_p
( seqid text, date_of_stop date not null, time_of_stop time, agency text
, subagency text, description text, location text, latitude numeric, longitude numeric
, accident text, belts Boolean, personal_injury Boolean, property_damage Boolean
, fatal Boolean, commercial_license Boolean, hazmat Boolean, commercial_vehicle Boolean
, alcohol Boolean, workzone boolean, state text, vehicletype text, year smallint
, make text, model text, color text, violation_type text, charge text, article text
, contributed_to_accident Boolean , race text, gender text, driver_city text
, driver_state text, dl_state text, arrest_type text , geolocation point
, council_districts smallint, councils smallint
, communities smallint, zip_codes smallint, municipalities smallint)
partition by range (date_of_stop);
```

Demo
PostgreSQL 10

Commands for the demo on PostgreSQL Version 10

```
/* Check the table structure */
\d traffic_violations_p

/* Create the table partitions */
create table traffic_violations_p_2012
partition of traffic_violations_p
for values from ('2012-01-01') to ('2013-01-01');

create table traffic_violations_p_2013
partition of traffic_violations_p
for values from ('2013-01-01') to ('2014-01-01');

create table traffic_violations_p_2014
partition of traffic_violations_p
for values from ('2014-01-01') to ('2015-01-01');
```

Commands for the demo on PostgreSQL Version 10

```
create table traffic_violations_p_2015
partition of traffic_violations_p
for values from ('2015-01-01') to ('2016-01-01');

create table traffic_violations_p_2016
partition of traffic_violations_p
for values from ('2016-01-01') to ('2017-01-01');

create table traffic_violations_p_2017
partition of traffic_violations_p
for values from ('2017-01-01') to ('2018-01-01');

create table traffic_violations_p_2018
partition of traffic_violations_p
for values from ('2018-01-01') to ('2019-01-01');
```

Demo
PostgreSQL 10

Commands for the demo on PostgreSQL Version 10

```
create table traffic_violations_p_2019
partition of traffic_violations_p
for values from ('2019-01-01') to ('2020-01-01');

create table traffic_violations_p_2020
partition of traffic_violations_p
for values from ('2020-01-01') to ('2021-01-01');

/* Insert the data */
insert into traffic_violations_p select * from mv_traffic_violations;

/* Insert data that does not fit into any partition
(won't work because of missing default partition) */
insert into traffic_violations_p (date_of_Stop) values ( now() );
```

Demo
PostgreSQL 10

Commands for the demo on PostgreSQL Version 10

```
/* Check the spreading in the partitions */

select count(*) from traffic_violations_p_2012;
select count(*) from traffic_violations_p_2013;
select count(*) from traffic_violations_p_2014;
select count(*) from traffic_violations_p_2015;
select count(*) from traffic_violations_p_2016;
select count(*) from traffic_violations_p_2017;
select count(*) from traffic_violations_p_2018;
select count(*) from traffic_violations_p_2019;
select count(*) from traffic_violations_p_2020;
```

Demo
PostgreSQL 10

Commands for the demo on PostgreSQL Version 10

```
/* As comparison for partition pruning in Version 11 check
the explain plans in Version 10 */

explain select count(*)
   from traffic_violations_p
   where date_of_stop = date('02-FEB-2013');

explain analyze
select count(*)
  from traffic_violations_p
 where date_of_stop = (select to_date('01.01.2014','DD.MM.YYYY'));
```

Demo
PostgreSQL 11

Commands for the demo on PostgreSQL Version 11

```
/* Create default partition */
create table traffic_violations_p_default partition of traffic_violations_p default;

/* Insert Data into Default partition */
insert into traffic_violations_p (date_of_Stop) values ( now() );

/* Check the data spreading */
select count(*) from traffic_violations_p_2012;
select count(*) from traffic_violations_p_2013;
select count(*) from traffic_violations_p_2014;
select count(*) from traffic_violations_p_2015;
select count(*) from traffic_violations_p_2016;
select count(*) from traffic_violations_p_2017;
select count(*) from traffic_violations_p_2018;
select count(*) from traffic_violations_p_2019;
select count(*) from traffic_violations_p_2020;
select count(*) from traffic_violations_p_default;
```

Demo
PostgreSQL 11

Commands for the demo on PostgreSQL Version 11

```
/* Create a table with Hash Partioning */
create table t_part_hash ( id int primary key, dummy text)
partition by hash (id);

/* Create the hash partitions */
create table t_part_hash_p1 partition of t_part_hash
for values with (modulus 5, remainder 0);
create table t_part_hash_p2 partition of t_part_hash
for values with (modulus 5, remainder 1);
create table t_part_hash_p3 partition of t_part_hash
for values with (modulus 5, remainder 2);
create table t_part_hash_p4 partition of t_part_hash
for values with (modulus 5, remainder 3);
create table t_part_hash_p5 partition of t_part_hash
for values with (modulus 5, remainder 4);
```

Demo
PostgreSQL 11

Commands for the demo on PostgreSQL Version 11

```
/* Insert data into hash table */
insert into t_part_hash (id, dummy)
       select a, md5(a::text) from generate_series(1,1000000) a;

/* Check data spreading across partitions */
select count(*) from t_part_hash_p1;
select count(*) from t_part_hash_p2;
select count(*) from t_part_hash_p3;
select count(*) from t_part_hash_p4;
select count(*) from t_part_hash_p5;

/* Try to create a default partition.
Won't work, because there is no need for a default partition*/
create table t_part_hash_default
partition of t_part_hash default;
```

Demo
PostgreSQL 11

Commands for the demo on PostgreSQL Version 11

```
/* Create an unique index */
create unique index traffic_violations_seqid_i on traffic_violations_p(seqid);

alter table traffic_violations_p add constraing traffic_violations_pk primary key
(seqid);

/* Create an index*/
create index traffic_violations_seqid_i on traffic_violations_p(seqid);

/* Create different indexes on different partitions */
create index traffic_violations_seqid_i1 on traffic_violations_p_2018(seqid);
create index traffic_violations_seqid_i2 on
traffic_violations_p_2019(seqid,date_of_stop);
```

Demo
PostgreSQL 11

Commands for the demo on PostgreSQL Version 11

```
/* Update the Date_of_stop column to check automatic data movement */

select count(*) from traffic_violations_p_2019;
select count(*) from traffic_violations_p_2020;

update traffic_violations_p set date_of_stop='2020-01-28' where date_of_stop='2019-01-
28';

select count(*) from traffic_violations_p_2019;
select count(*) from traffic_violations_p_2020;
```

Commands for the demo on PostgreSQL Version 11

```
/* Explain plans to see partition pruning in Version 11 */

explain select count(*)
   from traffic_violations_p
   where date_of_stop = date('02-FEB-2013');

explain analyze
select count(*)
  from traffic_violations_p
 where date_of_stop = (select to_date('01.01.2014','DD.MM.YYYY'));
```

Commands for the demo on PostgreSQL Version 12

```
/* create a table, create an index and create te partitions */
CREATE TABLE postleitzahl (ORT text, ZUSATZ char(50), PLZ int, VORWAHL int, BUNDESLAND
text) PARTITION BY RANGE (PLZ);
CREATE INDEX postleitzahl_i on postleitzahl(plz);


/* Create a table, an index, the partitions and subpartitions */
CREATE TABLE postleitzahl_0 partition of postleitzahl
     FOR VALUES FROM ('00001') to ('10000');
CREATE TABLE postleitzahl_2 partition of postleitzahl
     FOR VALUES FROM ('20001') to ('30000');
CREATE TABLE postleitzahl_3 partition of postleitzahl
     FOR VALUES FROM ('30001') to ('40000');
CREATE TABLE postleitzahl_4 partition of postleitzahl
     FOR VALUES FROM ('40001') to ('50000');
```

Commands for the demo on PostgreSQL Version 12

```
/* Create a table, an index, the partitions and subpartitions */
CREATE TABLE postleitzahl_5 partition of postleitzahl
     FOR VALUES FROM ('50001') to ('60000');
CREATE TABLE postleitzahl_6 partition of postleitzahl
     FOR VALUES FROM ('60001') to ('70000');
CREATE TABLE postleitzahl_7 partition of postleitzahl
     FOR VALUES FROM ('70001') to ('80000');
CREATE TABLE postleitzahl_8 partition of postleitzahl
     FOR VALUES FROM ('80001') to ('90000');
CREATE TABLE postleitzahl_9 partition of postleitzahl
     FOR VALUES FROM ('90001') to ('100000');
CREATE TABLE postleitzahl_default partition of postleitzahl default;
```

Commands for the demo on PostgreSQL Version 12

```
CREATE TABLE postleitzahl_1 partition of postleitzahl
     FOR VALUES FROM ('10001') to ('20000') partition by range (plz);

/* create two subpartitions */
CREATE TABLE postleitzahl_10_15 PARTITION OF postleitzahl_1
     FOR VALUES FROM ('10001') to ('15000');

CREATE TABLE postleitzahl_16_20 PARTITION OF postleitzahl_1
     FOR VALUES FROM ('15001') to ('20000');

/* Insert the data into the tables */
copy postleitzahl
from '/home/postgres/plz_de.csv'
delimiter ';'
CSV HEADER;
```

Commands for the demo on PostgreSQL Version 12

```
/* Select the data from the top table to the subpartition */
select count(*) from postleitzahl;
select count(*) from postleitzahl_0;
select count(*) from postleitzahl_1;
select count(*) from postleitzahl_2;
select count(*) from postleitzahl_3;
select count(*) from postleitzahl_4;
select count(*) from postleitzahl_5;
select count(*) from postleitzahl_6;
select count(*) from postleitzahl_7;
select count(*) from postleitzahl_8;
select count(*) from postleitzahl_9;
select count(*) from postleitzahl_default;
```

Demo
PostgreSQL 12

Commands for the demo on PostgreSQL Version 12

```
/* Use the new feature pg_partition_tree*/
SELECT * FROM pg_partition_tree('postleitzahl');
SELECT * FROM pg_partition_tree('postleitzahl_1');
SELECT * FROM pg_partition_tree('postleitzahl_2');
SELECT * FROM pg_partition_tree('postleitzahl_i');
```

Commands for the demo on PostgreSQL Version 12

```
/* Use the new feature pg_partition_root*/
select * FROM  pg_partition_root('postleitzahl_1');

/* Use the new feature pg_partition_ancestors*/
select * FROM  pg_partition_ancestors('postleitzahl_10_15');

/* Use the new feature \dP*/
\dP
\dP postleitzahl
\dP postleitzahl_i
```

Demo
PostgreSQL 13

Commands for the demo on PostgreSQL Version 13.1 and Version 13.2

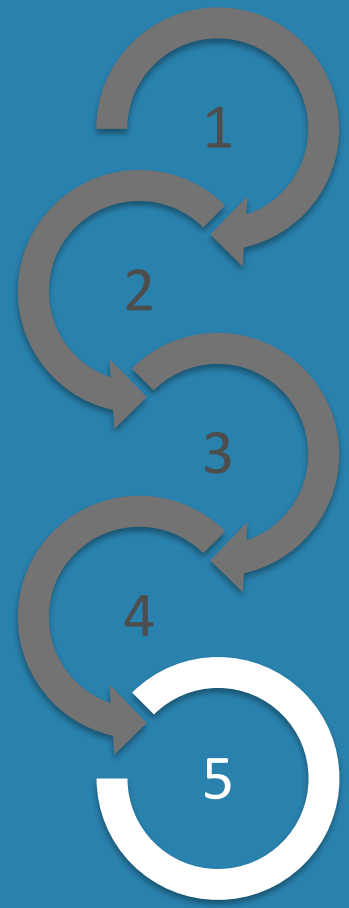
```
/* Create a tablespace */
CREATE TABLESPACE traffic
    LOCATION '/u02/pgdata/PG13/tablespace';

/* Create a partitioned table in the tablespace */
CREATE TABLE tablespace_test
( seqid text
, date_of_stop date not null)
PARTITION BY RANGE(date_of_stop)
tablespace traffic;

/* Try to delete the tablespace */
DROP TABLESPACE traffic;
```

Conclusion

- Current status
- Version 15 - development



Conclusion
Current status

A lot is already possible

> Range, list and hash partitioning
> Foreign keys
> Subpartitions
> Indexing and constraints
> Partition Pruning
> Attaching and detaching partitions

You miss something?

Conclusion
Version 15 - development

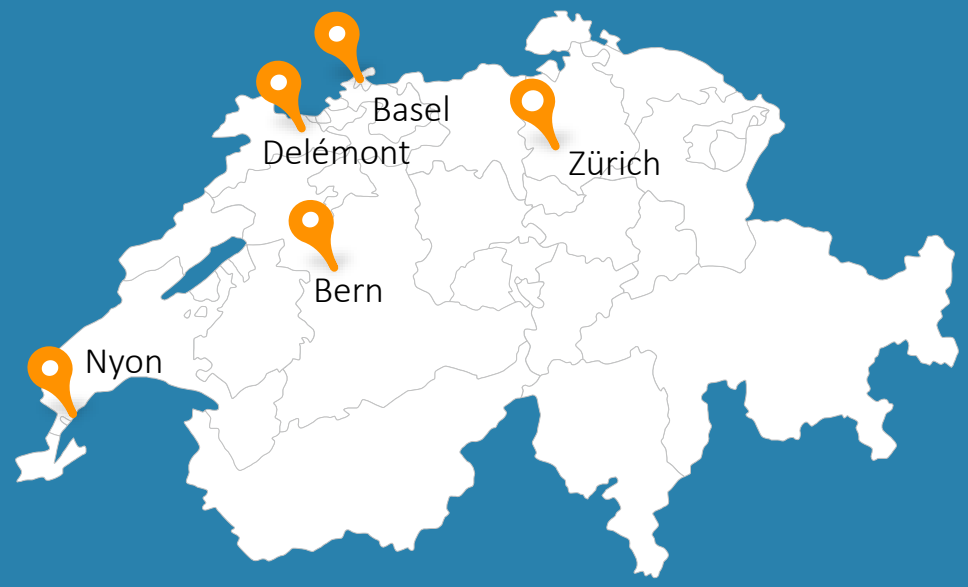
Already committed

> Foreign key triggers created for the partitioned tables and their partitions

> CLUSTER can now be executed on partitioned tables

Any questions?

Please do ask!



We would love to boost your IT-Infrastructure

How about you?