Table Partitioning in Postgres

Short Intro and What's New in v13

Amit Langote (@amitlan)

https://pgunconf.connpass.com/event/175483/

What is partitioning

- A syntax to define a hierarchy of tables that together store the data of a single logical relation.

```
CREATE TABLE foo (a int, b text);
```

VS.

```
CREATE TABLE foo (a int, b text) PARTITION BY RANGE (a);

CREATE TABLE foo_1 PARTITION OF foo FOR VALUES FROM (1) TO (1001);

CREATE TABLE foo_2 PARTITION OF foo FOR VALUES FROM (1001) TO (2001);
```

EDB POSTGRES

When to use partitioning

- Query performance worsening as tables and their indexes grow in size
  - Locality of access gets worse as data starts spreading over an ever expanding table, which means queries start to hit disk more often.
  - With smaller partitions each containing a related subset of rows, that is less likely.
  - Mainly a concern for OLTP workloads
- There is a requirement of bulk loading and deleting rows matching some condition
  - Load the data into a table, validate the condition using a CHECK constraint, use ATTACH PARTITION command to instantaneously make the table's data a part of the target table.
  - To perform bulk delete, simply drop partitions that match the condition
  - Typically seen in reporting workloads
- Postgres documentation has a good introduction and a great "Best Practices" section
  - https://www.postgresql.org/docs/12/ddl-partitioning.html

Timeline of Partitioning in Postgres

- Postgres 9.6 and earlier
  - Table inheritance, CHECK constraints, triggers
- Postgres 10
  - Syntax
- Postgres 11
  - Additional syntax: PARTITION BY HASH, DEFAULT partition
  - Creating indexes, triggers, primary key, foreign key on parent table
  - UPDATE can freely change partition key
  - Faster partition pruning, run-time partition pruning
  - Partition-level join, aggregation
- Postgres 12
  - Foreign keys referencing partitioned table
  - Performance: faster COPY, faster ordered scan of range partitions
  - Scalability for OLTP: fine to use many thousand partitions

New in Postgres 13

E.1.3.1. Server

E.1.3.1.1. Partitioning

- Improve cases where pruning of partitions can happen (Yuzuko Hosoya, Amit Langote, Álvaro Herrera)
- Allow partitionwise joins to happen in more cases (Ashutosh Bapat, Etsuro Fujita, Amit Langote, Tom Lane)

  For example, partitionwise joins can now happen between partitioned tables even when their partition bounds do not match exactly.
- Allow `BEFORE` row-level triggers on partitioned tables (Álvaro Herrera)

  These triggers cannot change which partition is the destination.
- Allow partitioned tables to be logically replicated via publications (Amit Langote)

  Previously, partitions had to be replicated individually. Now partitioned tables can be published explicitly causing all partitions to be automatically published. Addition/removal of partitions from partitioned tables are automatically added/removed from publications. The CREATE PUBLICATION option `publish_via_partition_root` controls whether changes to partitions are published as their own or their ancestor's.
- Allow logical replication into partitioned tables on subscribers (Amit Langote)

  Previously, subscribers could only receive rows into non-partitioned tables.
- Allow `ROW` values to be used as partitioning expressions (Amit Langote)

New in Postgres 13

- Improve partition pruning to better handle some corner cases

Postgres 12.2

```
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2)
partition by list (b);
create table foo2a partition of foo2 for values in ('a');
create table foo2def partition of foo2 default;

explain (costs off) select * from foo2 where a = 1;
        QUERY PLAN
---------------------------
 Append
   ->  Seq Scan on foo2a
         Filter: (a = 1)
   ->  Seq Scan on foo2def
         Filter: (a = 1)
(5 rows)
```

Postgres 13

```
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2)
partition by list (b);
create table foo2a partition of foo2 for values in ('a');
create table foo2def partition of foo2 default;

explain (costs off) select * from foo2 where a = 1;
        QUERY PLAN
---------------------------
 Result
   One-Time Filter: false
(2 rows)
```

New in Postgres 13

- Improve partition-wise join to handle more cases

Postgres 12.2

```
drop table foo,bar;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);
create table bar (a int) partition by list (a);
create table bar1 partition of bar for values in (1);
create table bar2 partition of bar for values in (2);
create table bar3 partition of bar for values in (3);
set enable_partitionwise_join to on;

explain (costs off) select * from foo join bar on foo.a = bar.a;
             QUERY PLAN
-----------------------------------
 Merge Join
   Merge Cond: (foo1.a = bar1.a)
   ->  Sort
         Sort Key: foo1.a
         ->  Append
               ->  Seq Scan on foo1
               ->  Seq Scan on foo2
   ->  Sort
         Sort Key: bar1.a
         ->  Append
               ->  Seq Scan on bar1
               ->  Seq Scan on bar2
               ->  Seq Scan on bar3
(13 rows)
```

Postgres 13

```
drop table foo,bar;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);
create table bar (a int) partition by list (a);
create table bar1 partition of bar for values in (1);
create table bar2 partition of bar for values in (2);
create table bar3 partition of bar for values in (3);

explain (costs off) select * from foo join bar on foo.a = bar.a;
                QUERY PLAN
------------------------------------------
 Append
   ->  Merge Join
         Merge Cond: (foo_1.a = bar_1.a)
         ->  Sort
               Sort Key: foo_1.a
               ->  Seq Scan on foo1 foo_1
         ->  Sort
               Sort Key: bar_1.a
               ->  Seq Scan on bar1 bar_1
   ->  Merge Join
         Merge Cond: (foo_2.a = bar_2.a)
         ->  Sort
               Sort Key: foo_2.a
               ->  Seq Scan on foo2 foo_2
         ->  Sort
               Sort Key: bar_2.a
               ->  Seq Scan on bar2 bar_2
(17 rows)
```

EDB POSTGRES

Postgres 12.2

```
set enable_partitionwise_join to on;
explain (costs off) select * from foo t1 full join foo t2 using (a) full join
foo t3 using (a);
                          QUERY PLAN
---------------------------------------------------------------
 Merge Full Join
   Merge Cond: (t3.a = (COALESCE(t1.a, t2.a)))
   ->  Sort
         Sort Key: t3.a
         ->  Append
               ->  Seq Scan on foo1 t3
               ->  Seq Scan on foo2 t3_1
   ->  Sort
         Sort Key: (COALESCE(t1.a, t2.a))
         ->  Result
               ->  Append
                     ->  Merge Full Join
                           Merge Cond: (t1.a = t2.a)
                           ->  Sort
                                 Sort Key: t1.a
                                 ->  Seq Scan on foo1 t1
                           ->  Sort
                                 Sort Key: t2.a
                                 ->  Seq Scan on foo1 t2
                     ->  Merge Full Join
                           Merge Cond: (t1_1.a = t2_1.a)
                           ->  Sort
                                 Sort Key: t1_1.a
                                 ->  Seq Scan on foo2 t1_1
                           ->  Sort
                                 Sort Key: t2_1.a
                                 ->  Seq Scan on foo2 t2_1
(27 rows)
```

Postgres 13

```
set enable_partitionwise_join to on;
explain (costs off) select * from foo t1 full join foo t2 using (a) full join
foo t3 using (a);
                          QUERY PLAN
---------------------------------------------------------------
 Append
   ->  Merge Full Join
         Merge Cond: (t3_1.a = (COALESCE(t1_1.a, t2_1.a)))
         ->  Sort
               Sort Key: t3_1.a
               ->  Seq Scan on foo1 t3_1
         ->  Sort
               Sort Key: (COALESCE(t1_1.a, t2_1.a))
               ->  Merge Full Join
                     Merge Cond: (t1_1.a = t2_1.a)
                     ->  Sort
                           Sort Key: t1_1.a
                           ->  Seq Scan on foo1 t1_1
                     ->  Sort
                           Sort Key: t2_1.a
                           ->  Seq Scan on foo1 t2_1
   ->  Merge Full Join
         Merge Cond: (t3_2.a = (COALESCE(t1_2.a, t2_2.a)))
         ->  Sort
               Sort Key: t3_2.a
               ->  Seq Scan on foo2 t3_2
         ->  Sort
               Sort Key: (COALESCE(t1_2.a, t2_2.a))
               ->  Merge Full Join
                     Merge Cond: (t1_2.a = t2_2.a)
                     ->  Sort
                           Sort Key: t1_2.a
                           ->  Seq Scan on foo2 t1_2
                     ->  Sort
                           Sort Key: t2_2.a
                           ->  Seq Scan on foo2 t2_2
(31 rows)
```

EDB POSTGRES

New in Postgres 13

- Allow BEFORE ROW triggers on partitioned tables
  - Trigger should not change the row to cause it to move to another partition

Postgres 12.2

```
drop table foo;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create function trigfunc() returns trigger language plpgsql as $$
begin new.a := new.a + 1; return new; end $$;

create trigger brtrig before insert on foo for each row execute
function trigfunc();
ERROR:  "foo" is a partitioned table
DETAIL:  Partitioned tables cannot have BEFORE / FOR EACH ROW
triggers.

create trigger brtrig before insert on foo1 for each row execute
function trigfunc();

insert into foo values (1);
ERROR:  new row for relation "foo1" violates partition constraint
DETAIL:  Failing row contains (2, null).
```

Postgres 13

```
drop table foo,bar;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create function trigfunc() returns trigger language plpgsql as $$
begin new.a := new.a + 1; return new; end $$;

create trigger brtrig before insert on foo for each row execute
function trigfunc();

insert into foo values (1);
ERROR:  moving row to another partition during a BEFORE FOR EACH
ROW trigger is not supported
DETAIL:  Before executing trigger "brtrig", the row was to be in
partition "public.foo1".
```

EDB POSTGRES

New in Postgres 13

- Allow partitioned tables to be directly replicated via publication
  - New publication parameter ‘publish_via_partition_root’

Postgres 12.2

```
drop table foo;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create publication pub for table foo;
ERROR:  "foo" is a partitioned table
DETAIL:  Adding partitioned tables to publications is not supported.
HINT:  You can add the table partitions individually.

create publication pub for table foo1, foo2;

create table foo3 partition of foo for values in (3);

\dRp+ pub
                      Publication pub
 Owner | All tables | Inserts | Updates | Deletes | Truncates
-------+------------+---------+---------+---------+-----------
 amit  | f          | t       | t       | t       | t
Tables:
    "public.foo1"
    "public.foo2"
```

Postgres 13

```
drop table foo,bar;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create publication pub for table foo;

\dRp+ pub
                              Publication pub
 Owner | All tables | Inserts | Updates | Deletes | Truncates | Via root
-------+------------+---------+---------+---------+-----------+----------
 amit  | f          | t       | t       | t       | t         | f
Tables:
    "Public.foo"

create publication pub_root for table foo with (publish_via_partition_root =
true);

\dRp+ pub_root
                              Publication pub_root
 Owner | All tables | Inserts | Updates | Deletes | Truncates | Via root
-------+------------+---------+---------+---------+-----------+----------
 amit  | f          | t       | t       | t       | t         | t
Tables:
    "public.foo"
```

EDB POSTGRES

New in Postgres 13

- Allow partitioned tables to receive changes via subscription

Postgres 12.2

```
drop table foo;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create subscription sub connection 'port=5432' publication pub;
ERROR:  cannot use relation "public.foo" as logical replication
target
DETAIL:  "public.foo" is a partitioned table.
```

Postgres 13

```
drop table foo;
create table foo (a int, b text) partition by list (a);
create table foo1 partition of foo for values in (1);
create table foo2 partition of foo for values in (2);

create subscription sub connection 'port=5432' publication pub;
create subscription sub_root connection 'port=5432' publication
pub_root;


select subname, string_agg(srrelid::regclass::text, ',') as tables
from pg_subscription s, pg_subscription_rel r where s.oid =
r.srsubid group by 1;
 subname  |  tables
----------+-----------
 sub_root | foo
 sub      | foo1,foo2
(2 rows)
```

- Wrote an article about this: https://amitlan.com/2020/05/14/partition-logical-replication.html

EDB POSTGRES

New in Postgres 13

- Allow ROW() expressions to be used as partitioning expressions

Postgres 12.2

```
drop table foo;
create table foo (a int, b text) partition by list
((foo));
ERROR:  partition key expressions cannot contain
whole-row references
```

Postgres 13

```
drop table foo;
create table foo (a int, b text) partition by list ((foo));
create table foo1 partition of foo for values in ('(1,1)');
create table foo2 partition of foo for values in ('(2,2)');

\d+ foo
                                  Partitioned table "public.foo"
 Column |  Type   | Collation | Nullable | Default | Storage  |
Stats target | Description
--------+---------+-----------+----------+---------+----------+-
-------------+-------------
 a      | integer |           |          |         | plain    |
|
 b      | text    |           |          |         | extended |
|
Partition key: LIST ((foo.*))
Partitions: foo1 FOR VALUES IN ('(1,1)'),
            foo2 FOR VALUES IN ('(2,2)')
```

EDB POSTGRES

Postgres 14 and beyond!

- Try to minimize the overhead of partitioning so it's more usable with, say, foreign keys

Both tables non-partitioned

```
alter table bar add foreign key (a) references foo;
Time: 6.107 ms

insert into foo select generate_series(1, 1000000);

insert into bar select generate_series(1, 1000000);
Time: 21898.402 ms (00:21.898)


truncate bar;
alter table bar drop constraint bar_a_fkey;
insert into bar select generate_series(1, 1000000);

alter table bar add foreign key (a) references foo;
Time: 1509.245 ms (00:01.509)
```

Both tables partitioned with 1000 partitions each

```
alter table bar add foreign key (a) references foo;
Time: 78174.651 ms (01:18.175)

insert into foo select generate_series(1, 1000000);

insert into bar select generate_series(1, 1000000);
Time: 160254.110 ms (02:40.254)


truncate bar;
alter table bar drop constraint bar_a_fkey;
insert into bar select generate_series(1, 1000000);

alter table bar add foreign key (a) references foo;
Time: 479498.926 ms (07:59.499)
```

EDB POSTGRES