PostgreSQL partitioning. Work in progress

Anastasia Lubennikova, Sr. Database Developer
February 6, FOSDEM 2021

www.postgrespro.com

Objectives

- Learn more about partitioning
- Collect useful resources for hackers
- Compare native PostgreSQL partitioning and third-party solutions: pg_pathman and pg_partman.
- Clarify our roadmap for partitioning
- Provide an overview for features in progress and encourage people to review them.

Disclaimer: The information included in this talk is relevant as of January 15, 2021.

Declarative partitioning

- A set of features for handling large data volumes.

  It includes dedicated syntax for partitioning,

  convenient data management options and internal performance optimizations

Usability (Partition Maintenance) + Performance + Integration with other features

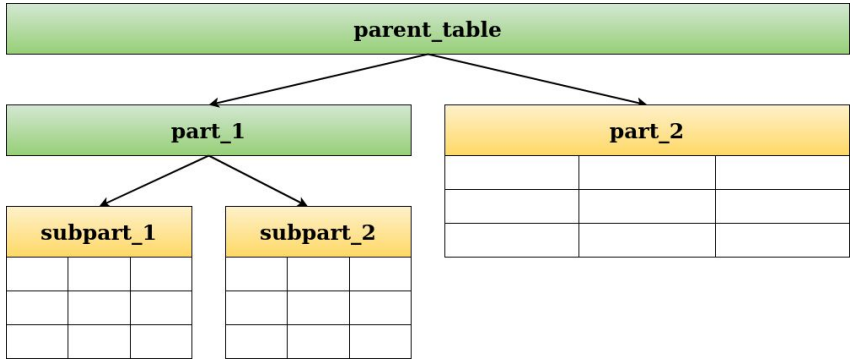
Declarative partitioning

- Limited declarative partitioning was introduced in PostgreSQL 10.
- In PostgreSQL 11, many usability features were implemented.
- PostgreSQL 12 significantly improved the performance of queries over partitioned data.
- PostgreSQL 13 improved performance and integration with other PostgreSQL features, such as row-level BEFORE triggers and logical replication.

For more information, please check the Table Partitioning in Postgres: How Far We've Come talk by Amit Langote.

Declarative partitioning

- The partitioned table & index are always empty (i.e. don’t have a physical storage).
- **Leaf** tables - the ones that contain the data.
- A partitioned table is defined by the **partitioning strategy** and **partition key**.
  - Partitioning strategies are HASH, LIST, RANGE.
  - Partition key - a set of columns or expressions.
- The subset of data in a partition is defined by its **partition bounds**.
- A **default** partition stores rows that do not belong to any existing partition.



System catalogs

- The **partition strategy** and **key** are stored in the **pg_partitioned_table** system catalog.

```
select * from pg_partitioned_table;
partrelid | partstrat | partnatts | partdefid | partattrs | partclass | partcollation | partexprs
-----------+-----------+-----------+-----------+-----------+-----------+---------------+-----------
     16392 | r         |         1 |         0 | 2         | 3127      | 0             |
```

- The **partition bounds** are stored in the **pg_class.relpartbound** system catalog attribute.

  The **pg_class.relispartition** flag is true if the table or index is a partition.

```
select relname, relpartbound from pg_class;
relname      | part_1
relpartbound | {PARTITIONBOUNDSPEC :strategy r :is_default false :modulus 0 :remainder 0 :listdatums <>
:lowerdatums ({PARTITIONRANGEDATUM :kind -1 :value <> :location 63}) :upperdatums ({PARTITIONRANGEDATUM
:kind 0 :value {CONST :consttype 1184 :consttypmod -1 :constcollid 0 :constlen 8 :constbyval true
:constisnull false :location 77 :constvalue 8 [ 0 -96 -83 48 -54 90 2 0 ]} :location 77}) :location 57}
```

Tuple routing

- The set of partition columns and their names and types must exactly match the parental ones.
- The existing tables can be ATTACHed to the target table or DETACHed from it as partitions.
- It is not possible to turn a regular table into a partitioned table or vice versa.
- Overlapping partition bounds are not allowed.
- Default values may be specified separately for each partition.
  Though, they are not applied when inserting a tuple through a partitioned table.

Tuple routing

- CHECK and NOT NULL constraints of a partitioned table are always inherited by all its partitions.
- There is no way to enforce cross-partition restrictions.

  → Unique constraints must include all partition key columns.

  → There’s no way to create a global exclusion constraint spanning the whole partitioned table.

Strategy-specific details

- HASH partitioning doesn’t allow setting the default partition.
- LIST partitioning doesn’t allow a multicolumn partition key.
- RANGE partition bounds may take special values: MINVALUE and MAXVALUE.
- A non-default RANGE partition table does not currently allow partition keys to be NULL.

Locks used for partitioning

- Dropping a partition requires an AccessExclusiveLock on the parent table.
- DETACH PARTITION requires an AccessExclusiveLock on the parent table.
- Concurrent index builds on partitioned tables are currently not supported.
- ATTACH PARTITION requires an AccessExclusiveLock on the default partition.

  The default partition must be scanned to verify that it does not contain any rows not properly belonging in the new one.

Usability features. TODO

- Generate bounds automatically
- Define a callback (trigger) for partition creation event
- Change partition bounds
- Split / merge partitions [CONCURRENTLY]
- Partition / re-partition existing table [CONCURRENTLY]
- Global indexes

Partitioning. Third-party tools

- pg_pathman https://github.com/postgrespro/pg_pathman
  - extension developed & maintained by Postgres Professional
  - PostgreSQL >= 9.5. Now deprecated in favor of PostgreSQL's native declarative partitioning.
  - HASH, RANGE partitioning.

- pg_partman https://github.com/pgpartman/pg_partman
  - extension developed & maintained by Crunchy Data.
  - PostgreSQL >= 9.5
  - Time or serial id partitioning.
  - Trigger-based for older versions, native-based for PG10+

pg_pathman features

- pg_pathman provides an optimized partitioning mechanism
  and functions to manage partitions.

  - Partition management (attach, detach, add, drop, split, merge, replace).
  - Automatic partition creation for RANGE partitioning.
  - User-defined callbacks for partition creation event handling.
  - Non-blocking concurrent table partitioning.
  - Optimized planning mechanism: runtime pruning, partitionwise joins & aggregates.

pg_partman features

- pg_partman - an extension that simplifies managing of time or serial id based table partitioning.
  - Automated creation of partitions.
  - Partitioning of the existing table (+ batch mode for better concurrency).
  - Move data from child tables back to the parent table.
  - BG worker to run partition maintenance: auto creation, retention.

Third-party tools. Pros & cons

+ Side-project is easier at the start as it needs less compromises.
+ More often releases → more feedback from users.
+ Not so strict quality requirements :)

- Not everything can be done from an extension.
- Difficult to maintain.
- Not available in the clouds such as Amazon RDS.

Partitioning. Work in progress

- Automatic HASH and LIST partition creation

  Bound generation for HASH and LIST is pretty straightforward.

  RANGE needs more work, but it is doable. First of all, we need to agree on a new syntax which needs to be extendable for future improvements.

- CLUSTER on partitioned table

  Perform CLUSTER automatically from the top-most parent for any partitions that have a suitable index.

- Autovacuum on partitioned tables

  Fix autovacuum analyze on partitioned tables, so that their statistics were updated correctly.

Partitioning. Work in progress

- ALTER TABLE .. DETACH PARTITION CONCURRENTLY

  Detach partitions without blocking concurrent activity.

- DROP INDEX CONCURRENTLY on partitioned index

  CREATE INDEX CONCURRENTLY on partitioned table

  Impose fewer locking requirements on index maintenance.

Partitioning. Gotchas

- We need to agree on a **new syntax** for new features.
- To provide **partition maintenance** … we need to add a trigger or a callback on partition creation. Event triggers are not detailed enough to handle this.
- Object naming **length limit** of 63 characters makes partition name generation quite difficult.

Links

- Table partitioning documentation.
- README about partitionwise joins and aggregates.
- Table Partitioning in Postgres: How Far We've Come talk by Amit Langote.
- Table partitioning page on the PostgreSQL Wiki (slightly outdated).
- Proposal of declarative partitioning improvements on the PostgreSQL Wiki.

Postgres PROFESSIONAL

Thank you for attention!
Any questions?

a.lubennikova@postgrespro.com

https://postgrespro.com/