Practical Partitioning in Production with Postgres

Jimmy Angelakos
Senior PostgreSQL Architect

Postgres Vision 2021-06-23
We’ll be looking at:

• Intro to Partitioning in PostgreSQL
• Why?
• How?
• Practical Example
Introduction to Partitioning in PostgreSQL
What is partitioning?

- RDBMS context: division of a table into distinct independent tables
- Horizontal partitioning (by row) – different rows in different tables
- Why?
  - Easier to manage
  - Performance
Partitioning in PostgreSQL

HISTORY

• Has had partitioning for quite some time now… PG 8.1 (2005)
  – Inheritance-based
  – Why haven’t I heard of this before?
  – It’s not great tbh...
• Declarative Partitioning: PG 10 (2017)
  – Massive improvement
Declarative Partitioning

(PG 10+)

Specification of:

• Partitioning method
• Partition key
  - Column(s) or expression(s)
  - Value determines data routing
• Partition boundaries

By declaring a table (DDL):

```
CREATE TABLE cust (id INT, signup DATE)
PARTITION BY RANGE (signup);
```

```
CREATE TABLE cust_2020
PARTITION OF cust FOR VALUES FROM ('2020-01-01') TO ('2021-01-01');
```

• Partitions may be partitioned themselves (sub-partitioning)
Why?
PostgreSQL limits

(Hard limits, hard to reach)

- Database size: unlimited
- Tables per database: 1.4 billion
- Table size: 32 TB
  - Default block size: 8192 bytes
- Rows per table: depends
  - As many as can fit onto 4.2 billion blocks
What partitioning can help with (i)

(Very large tables)

• Disk size limitations
  − You can put partitions on different tablespaces

• Performance
  − Partition pruning
  − Table scans
  − Index scans
  − Hidden pitfalls of very large tables*
What partitioning can help with (ii)

(Very large tables)

• Maintenance
  - Deletions (some filesystems are bad at deleting large numbers of files)
  - `DROP TABLE cust_2020;`
  - `ALTER TABLE cust DETACH PARTITION cust_2020;`

• VACUUM
  - Bloat
  - Freezing → `xid` wraparound
What partitioning is not

• Magic bullet
  − No substitute for rational database design

• Sharding
  − Not about putting part of the data on different nodes

• Performance tuning
  − Unless you have one of the mentioned issues
How?
Dimensioning

Plan ahead!

• Get your calculator out
  - Data ingestion rate (both rows and size in bytes)
  - Projected increases (e.g. 25 locations projected to be 200 by end of year)
  - Data retention requirements
• Will inform choice of partitioning method and key
• For instance: 1440 measurements/day from each of 1000 sensors – extrapolate per year
• Keep checking if this is valid and be prepared to revise
Partitioning method

Dimensioning usually makes this clearer

- **Range**: For key column(s) e.g. ranges of dates, identifiers, etc.
  - Lower end: inclusive, upper end: exclusive
- **List**: Explicit key values stated for each partition
- **Hash (PG 11+)**: If you have a column with values close to unique
  - Define Modulus ( & remainder ) for number of almost-evenly-sized partitions
Partition Key selection

Choose wisely - know your data!

• Analysis
  - Determine main keys used for retrieval from queries
  - Proper key selection enables **partition pruning**
  - Can use multiple columns for higher granularity (more partitions)

• Desirable
  - High enough cardinality (range of values) for the number of partitions needed
  - A column that doesn’t change often, to avoid moving rows among partitions
Sub-partitioning

• Simply put, partitions are partitioned tables themselves. Plan ahead!

• **CREATE TABLE** transactions (…, location_code TEXT, tstamp TIMESTAMPTZ)  
  PARTITION BY RANGE (tstamp);

• **CREATE TABLE** transactions_2021_06  
  PARTITION OF transactions FOR VALUES FROM ('2021-06-01') TO ('2021-07-01')  
  PARTITION BY HASH (location_code);

• **CREATE TABLE** transactions_2021_06_p1  
  PARTITION OF transactions_2021_06 FOR VALUES WITH (MODULUS 4, REMAINDER 0);
Partitioning by multiple columns

Be careful!

- CREATE TABLE transactions (... , location_code TEXT, tstamp TIMESTAMPTZ) PARTITION BY RANGE (tstamp, location_code);
- CREATE TABLE transactions_2021_06_a PARTITION OF transactions FOR VALUES FROM ('2021-06-01', 'AAA') TO ('2021-07-01', 'AZZ');
- CREATE TABLE transactions_2021_06_b PARTITION OF transactions FOR VALUES FROM ('2021-06-01', 'BAA') TO ('2021-07-01', 'BZZ');
  ERROR: partition "transactions_2021_06_b" would overlap partition "transactions_2021_06_a"

- Because tstamp '2021-06-01' can only go in the first partition!
What Postgres does not do

• Automatic creation of partitions
  – Create in advance
  – Use a cronjob

• Imperative merging/splitting of partitions
  – Move rows manually

• Sharding to different nodes
  – You may have to configure FDW manually
Practical Example
Partitioning a live production system

• Is your table too large to handle?
• Can partitioning help?
• What if it’s in constant use?
The situation

Huge 20 TB table

• OLTP workload, transactions keep flowing in
  – Table keeps increasing in size
• VACUUM never ends
  – Has been running for a full month already…
• Queries are getting slower
  – Not just because of sheer number of rows...
* Hidden performance pitfall (i)

For VERY large tables

- Postgres has 1GB **segment size**
  - Can only be changed at compilation time
  - 20 TB table = 20000 segments (files on disk)

- Why is this a problem?
  - `md.c` →
* Hidden performance pitfall (ii)

- This loops 20000 times every time you want to access a table page
  - Linked list of segments
- Code from PG 9.6
- It has been heavily optimised recently (caching, etc).
- Still needs to run a lot of times

```c
1973 /*
1974 * Get number of blocks present in a single disk file
1975 */
1976 static BlockNumber
1977 _mndbblocks(MgrRelation reln, ForkNumber forknum, MdfdVec *seg)
1978 {
1979     off_t      len;
1980
1981     len = FileSeek(seg->mdfd_vfd, 0L, SEEK_END);
1982     if (len < 0)
1983         ereport(ERROR,
1984             (errcode_for_file_access(),
1985             errmsg("could not seek to end of file \"%s\": %m",
1986             FilePathName(seg->mdfd_vfd))));
1987 /* note that this calculation will ignore any partial block at EDF */
1988     return (BlockNumber) (len / BLCKSZ);
1989 }
```
So what do we do?

Next steps

• Need to partition the huge table
  – Dimensioning
  – Partition method
  – Partition key

• Make sure we’re on the latest version (PG 13)
  – Get latest features & performance enhancements
What is our table like?

It holds daily transaction totals for each point of sales

• Dimensioning
  - One partition per month will be about 30GB of data, so acceptable size

• Method, Key
  - Candidate key is transaction date, which we can partition by range
  - Check that there are no data errors (e.g. dates in the future when they shouldn’t be)

• Partition sizes don’t have to be equal
  - We can partition older, less often accessed data by year
Problems

What things you cannot do in production

• Lock the table totally (**ACCESS EXCLUSIVE**) or prevent writes
  – People will start yelling, and they will be right

• Cause excessive load on the system (e.g. I/O) or cause excessive disk space usage
  – Can’t copy whole 20 TB table into empty partitioned table
  – See above about yelling

• Present an inconsistent or incomplete view of the data
The plan

Take it step by step

• Rename the huge table and its indices
• Create an empty partitioned table with the old huge table’s name
• Create the required indices on the new partitioned table
  – They will be created automatically for each new partition
• Create first new partition for new incoming data
• Attach the old table as a partition of the new table so it can be used normally*
• Move data out of the old table incrementally at our own pace
Rename the huge table and its indices

-- Do this all in one transaction
BEGIN;

ALTER TABLE dailytotals RENAME TO dailytotals_legacy;

ALTER INDEX dailytotals_batchid RENAME TO dailytotals_legacy_batchid;

ALTER INDEX ...
...

© Copyright EnterpriseDB Corporation, 2021. All rights reserved.
CREATE TABLE dailytotals (  
totalid BIGINT NOT NULL DEFAULT nextval('dailytotals_totalid_seq')  
, totaldate DATE NOT NULL  
, totalsum BIGINT  
...  
, batchid BIGINT NOT NULL  
)  
PARTITION BY RANGE (totaldate);  

CREATE INDEX dailytotals_batchid ON dailytotals (batchid);  
...
Create partition for new incoming data

```sql
CREATE TABLE dailytotals_202106
PARTITION OF dailytotals
FOR VALUES FROM ('2021-06-01') TO ('2021-07-01');
```
DO $$
DECLARE earliest DATE;
DECLARE latest DATE;
BEGIN

-- Set boundaries
SELECT min(totaldate) INTO earliest FROM dailytotals_legacy;
latest := '2021-06-01'::DATE;

Attach old table as a partition (i)
Attach old table as a partition (ii)

-- HACK HACK HACK (only because we know and trust our data)
ALTER TABLE dailytotals_legacy
ADD CONSTRAINT dailytotals_legacy_totaldate
CHECK (totaldate >= earliest AND totaldate < latest)
NOT VALID;

-- You should not touch pg_catalog directly
UPDATE pg_constraint
SET convalidated = true
WHERE conname = 'dailytotals_legacy_totaldate';
ALTER TABLE dailytotals
ATTACH PARTITION dailytotals_legacy
FOR VALUES FROM (earliest) TO (latest);

END;
$$ LANGUAGE PLPGSQL;
COMMIT;
Move data from old table at our own pace

- For instance, during quiet hours for the system, in scheduled batch jobs, etc.

```sql
WITH rows AS (
  DELETE FROM dailytotals_legacy d
  WHERE (totaldate >= '2020-01-01' AND totaldate < '2021-01-01')
  RETURNING d.* )
INSERT INTO dailytotals SELECT * FROM rows;
```

- In the same transaction: `DETACH` the old table, perform the move, `reATTACH` with changed boundaries. Rinse and repeat!
- Make sure the target partition exists!
Partitioning improvements

Make sure you’re on the latest release so you have them!

• **PG11:** DEFAULT partition, **UPDATE** on partition key, **HASH** method, PKs, Fks, Indexes, Triggers

• **PG12:** Performance (pruning, **COPY**), FK references for partitioned tables, ordered scans

• **PG13:** Logical replication for partitioned tables, improved performance (**JOINs**, pruning)

• (Soon) **PG14:** **REINDEX CONCURRENTLY**, **DETACH CONCURRENTLY**, faster **UPDATE/DELETE**
To conclude...

- Know your data!
- Upgrade – be on the latest release!
- Partition before you get in deep water!
- Find me on Twitter: @vyruss