pganalyze

Mastering PostgreSQL Partitioning: Advanced Questions and Demos

Postgres Conference San Jose 2026

Ryan Booz

Ryan Booz

Solutions Engineer
pganalyze

@ryanbooz

/in/ryanbooz

www.softwareandbooz.com

youtube.com/@ryanbooz

github.com/ryanbooz

Agenda

Partitioning Review

Advanced Topics

Improvements in Recent Releases

Partitioning Review

What Are Partitions?

- Regular PostgreSQL tables
  - Schema
  - Tablespace
- Attached to a parent table (children)
- Self-contained indexes
- Can also be parent tables (sub-partitioning)

Declarative Partitioning

pganalyze

- PostgreSQL 10+
- Identify partitioning key column
- Specify method (RANGE, LIST, HASH)
- Partitions must be created before data arrives
- Default Partition = catch all
    - Can introduce challenging maintenance

Data Retention/Archiving

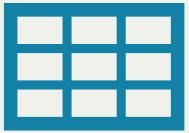
- Without partitions, DELETE adds significant overhead
  - MVCC bloat
  - Index maintenance
  - Potential blocking
- With partitions
  - DETACH PARTITION
  - DROP TABLE

Primary keys and unique constraints

All unique constraints **must** be included in the **partitioning key**

Range

Time-series (logdate)


[Jan 1 – Feb 1)


[Feb 1 – Mar 1)


[Mar 1 – Apr 1)

Numeric (category_id)


[1-10)


[10-20)


[20-30)

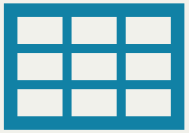
```
CREATE TABLE bluebox.payment (
  payment_id integer GENERATED BY DEFAULT
      AS IDENTITY PRIMARY KEY,
  customer_id integer NOT NULL,
  rental_id integer NOT NULL,
  amount numeric(5,2) NOT NULL,
  payment_date timestamp with time zone NOT NULL
);
```

```
CREATE TABLE bluebox.payment2 (
  payment_id serial4 NOT NULL,
  ...
  payment_date timestamptz NOT NULL,
  CONSTRAINT payment_bak_pkey
       PRIMARY KEY (payment_date, payment_id)
)
PARTITION BY RANGE (payment_date);

CREATE TABLE bluebox.payment2_y2024m01
       PARTITION OF bluebox.payment2
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');

CREATE TABLE bluebox.payment2_y2024m02
       PARTITION OF bluebox.payment2
FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');
```

List

Partition on “state” column

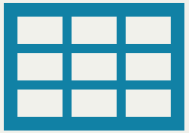
(PA,VA,NY,NJ)



(VA,WV,NH,MA)

List – Modify by addition

Partition on “state” column



(PA,VA,NY,NJ)

(VA,WV,NH,MA)

(RI,ME,VT,CT)

```
CREATE TABLE bluebox.customer (
  customer_id int8 NOT NULL,
  store_id int4 NOT NULL,
  full_name text NOT NULL,
  email text NULL,
  ...,
  state text NOT NULL,
  ...
)
PARTITION BY LIST (state);

CREATE TABLE bluebox.customer_northeast
      PARTITION OF bluebox.customer
FOR VALUES IN ('NY','PA','VT','CT','RI','ME','NH','NJ');
```

Hash

- Specify a MODULUS and REMAINDER
- No clear partitioning key
- Generally even distribution of data
- Cannot re-partition in the future without rewrite

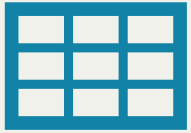
Hash

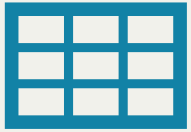
payment_id


(modulus 8, remainder 0)


(modulus 8, remainder 1)


(modulus 8, remainder 2)


(...)


(modulus 8, remainder 7)

```
CREATE TABLE bluebox.payment2 (
  rental_id int NOT NULL,
  ...
  payment_date timestamptz NOT NULL,
  CONSTRAINT payment_bak_pkey
       PRIMARY KEY (rental_id)
)
PARTITION BY HASH (rental_id);

CREATE TABLE bluebox.payment2_p0
       PARTITION OF bluebox.payment2
FOR VALUES WITH (MODULUS 8, REMAINDER 0);

CREATE TABLE bluebox.payment2_p1
       PARTITION OF bluebox.payment2
FOR VALUES WITH (MODULUS 8, REMAINDER 1);
```

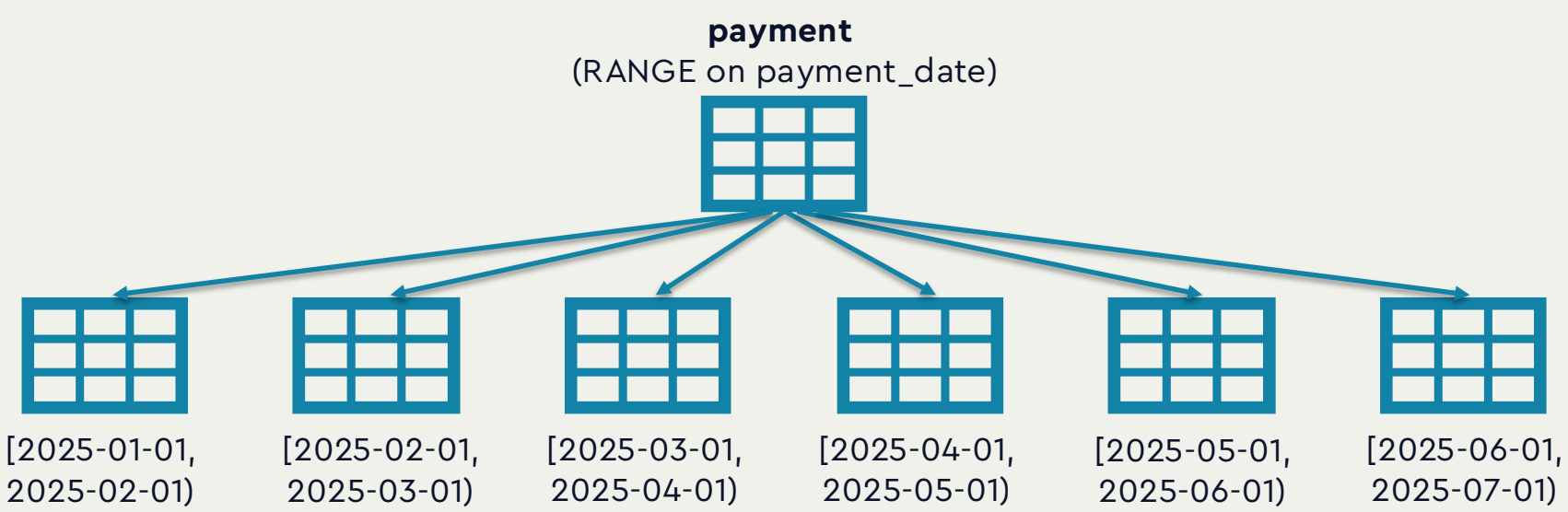
Advanced Topics

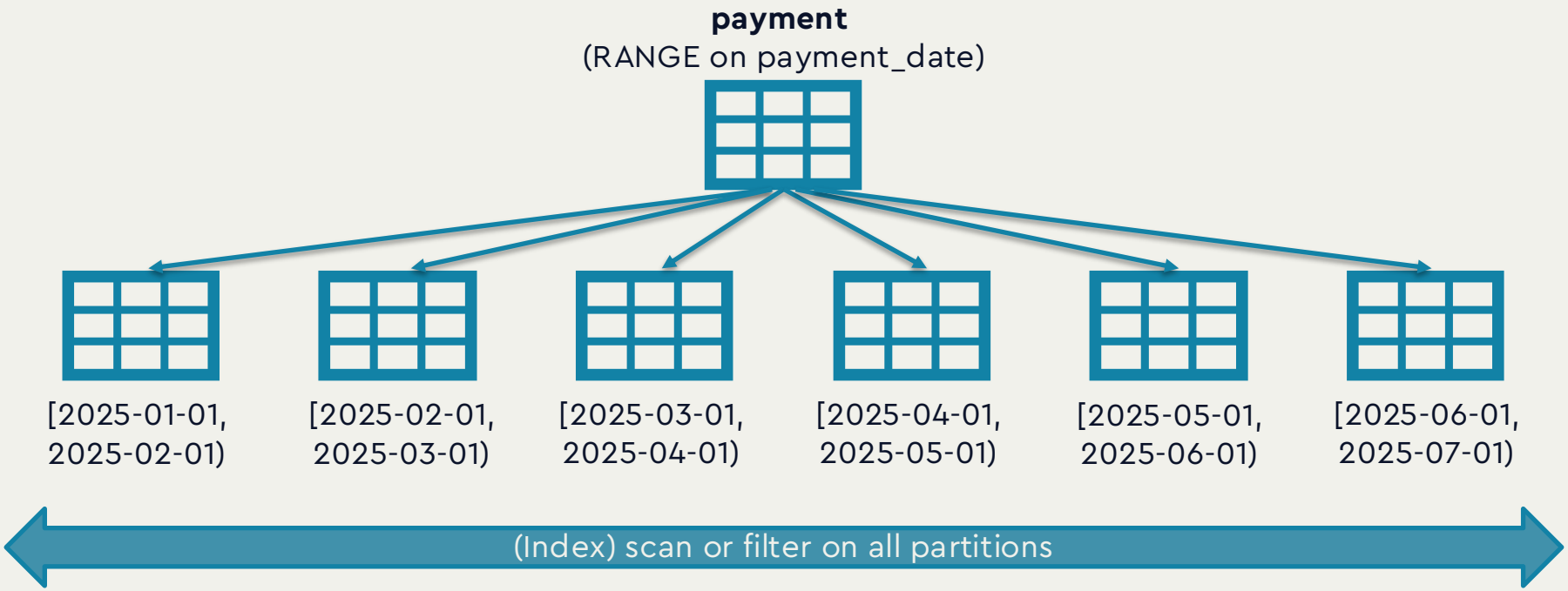
Efficient Query Partition Pruning and Sub-partitions

Partition pruning on non-partitioned columns

pganalyze

- RANGE partitions by date are helpful for efficient date queries and natural archiving boundaries
- They do not encourage efficient partitioning on alternate/primary keys


payment
(RANGE on payment_date)
[2025-01-01, 2025-02-01)
[2025-02-01, 2025-03-01)
[2025-03-01, 2025-04-01)
[2025-04-01, 2025-05-01)
[2025-05-01, 2025-06-01)
[2025-06-01, 2025-07-01)




```
SELECT * FROM payment2 p
 JOIN rental r USING (rental_id)
WHERE rental_id = 20042391;
```

```
QUERY PLAN                                                                                                                                                   |
-------------------------------------------------------------------------------------------------------------------------------------------------------------+
Nested Loop  (cost=4.63..549.77 rows=178 width=80) (actual time=0.187..0.232 rows=1 loops=1)                                                                 |
  Buffers: shared hit=105                                                                                                                                    |
  ->  Index Scan using rental_pkey on rental r  (cost=0.43..8.45 rows=1 width=49) (actual time=0.019..0.020 rows=1 loops=1)                                  |
        Index Cond: (rental_id = 20042391)                                                                                                                   |
        Buffers: shared hit=4                                                                                                                                |
  ->  Append  (cost=4.20..539.55 rows=178 width=35) (actual time=0.160..0.204 rows=1 loops=1)                                                                |
        Buffers: shared hit=101                                                                                                                              |
        ->  Bitmap Heap Scan on payment2_202301 p_1  (cost=4.20..13.67 rows=6 width=36) (actual time=0.007..0.007 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202301_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.002..0.003 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202302 p_2  (cost=4.20..13.67 rows=6 width=36) (actual time=0.002..0.002 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202302_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.001 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202303 p_3  (cost=4.20..13.67 rows=6 width=36) (actual time=0.003..0.003 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202303_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.001 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202304 p_4  (cost=4.20..13.67 rows=6 width=36) (actual time=0.002..0.003 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202304_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.001 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202305 p_5  (cost=4.20..13.67 rows=6 width=36) (actual time=0.003..0.003 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202305_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.001 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202306 p_6  (cost=4.20..13.67 rows=6 width=36) (actual time=0.003..0.003 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202306_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.001 rows=0 loops=1)             |
                    Index Cond: (rental_id = 20042391)                                                                                                       |
                    Buffers: shared hit=2                                                                                                                    |
        ->  Bitmap Heap Scan on payment2_202307 p_7  (cost=4.20..13.67 rows=6 width=36) (actual time=0.003..0.003 rows=0 loops=1)                            |
              Recheck Cond: (rental_id = 20042391)                                                                                                           |
              Buffers: shared hit=2                                                                                                                          |
              ->  Bitmap Index Scan on payment2_202307_rental_id_idx  (cost=0.00..4.20 rows=6 width=0) (actual time=0.001..0.002 rows=0 loops=1)             |
```

Partition pruning on non-partitioned columns

pganalyze

- All partitions are planned and queried
- Query execution can greatly improve with an index on the filtering column (reduced overall execution)
- Still – a lot of data access for no value

```
->  Index Scan using payment2_202601_rental_id_idx on payment2_202601 p_37  (cost=0.29..8.31 rows=1 width=31) (actual time=0.005..0.005
rows=0 loops=1)|
              Index Cond: (rental_id = 20042391)
|
              Buffers: shared hit=2
|
```

-> Index Scan using payment2_202602_rental_id_idx on payment2_202602 p_38 (cost=0.29..8.31 rows=1 width=31) (actual time=0.009..0.010 rows=1 loops=1)|
Index Cond: (rental_id = 20042391)

```
|
              Buffers: shared hit=3
|
        ->  Index Scan using payment2_202603_rental_id_idx on payment2_202603 p_39  (cost=0.42..8.44 rows=1 width=31) (actual
time=0.005..0.005 rows=0 loops=1)|
              Index Cond: (rental_id = 20042391)
|
              Buffers: shared hit=3
|
        ->  Index Scan using payment2_202604_rental_id_idx on payment2_202604 p_40  (cost=0.42..8.44 rows=1 width=31) (actual
time=0.005..0.005 rows=0 loops=1)|
              Index Cond: (rental_id = 20042391)
|
```

```
Planning:
  Buffers: shared hit=78
Planning Time: 1.566 ms
Execution Time: 0.421 ms
```

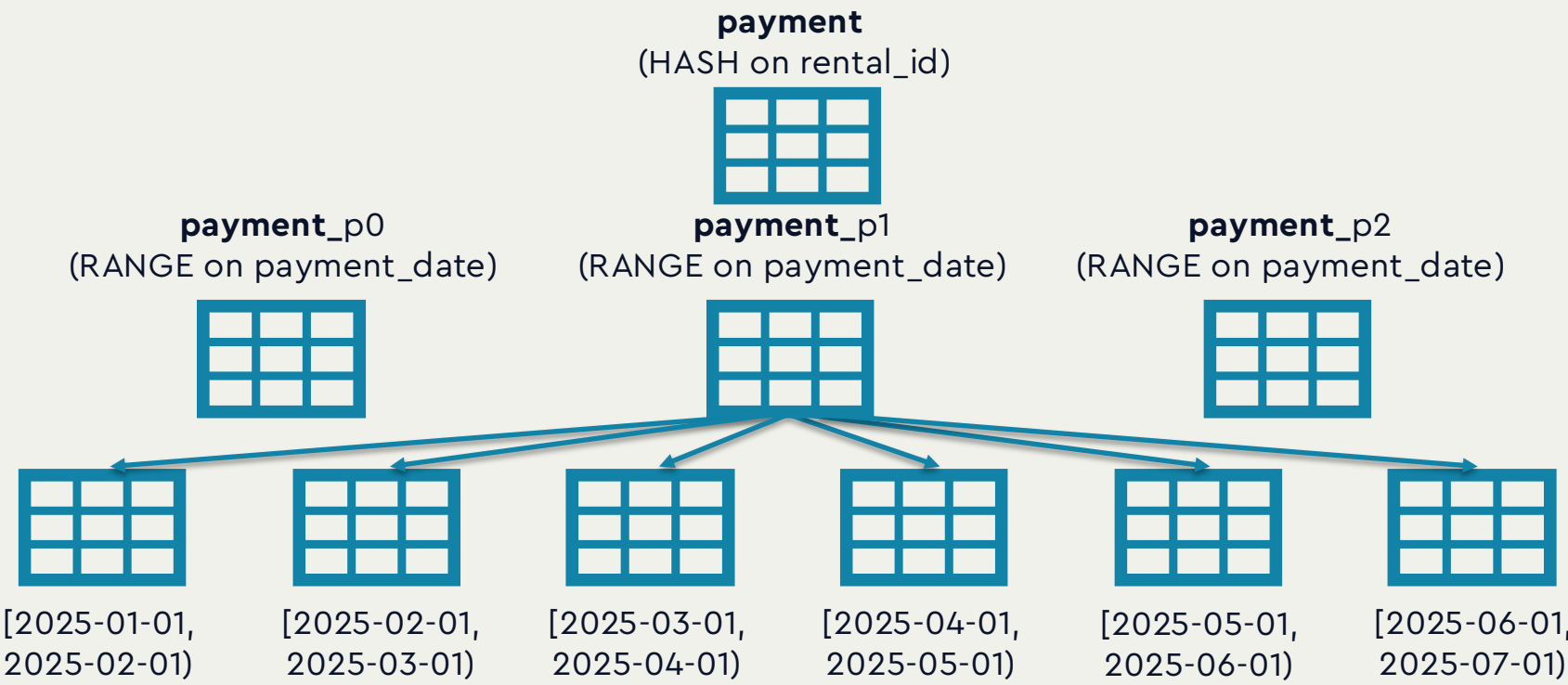
Sub-partitioning for better locality

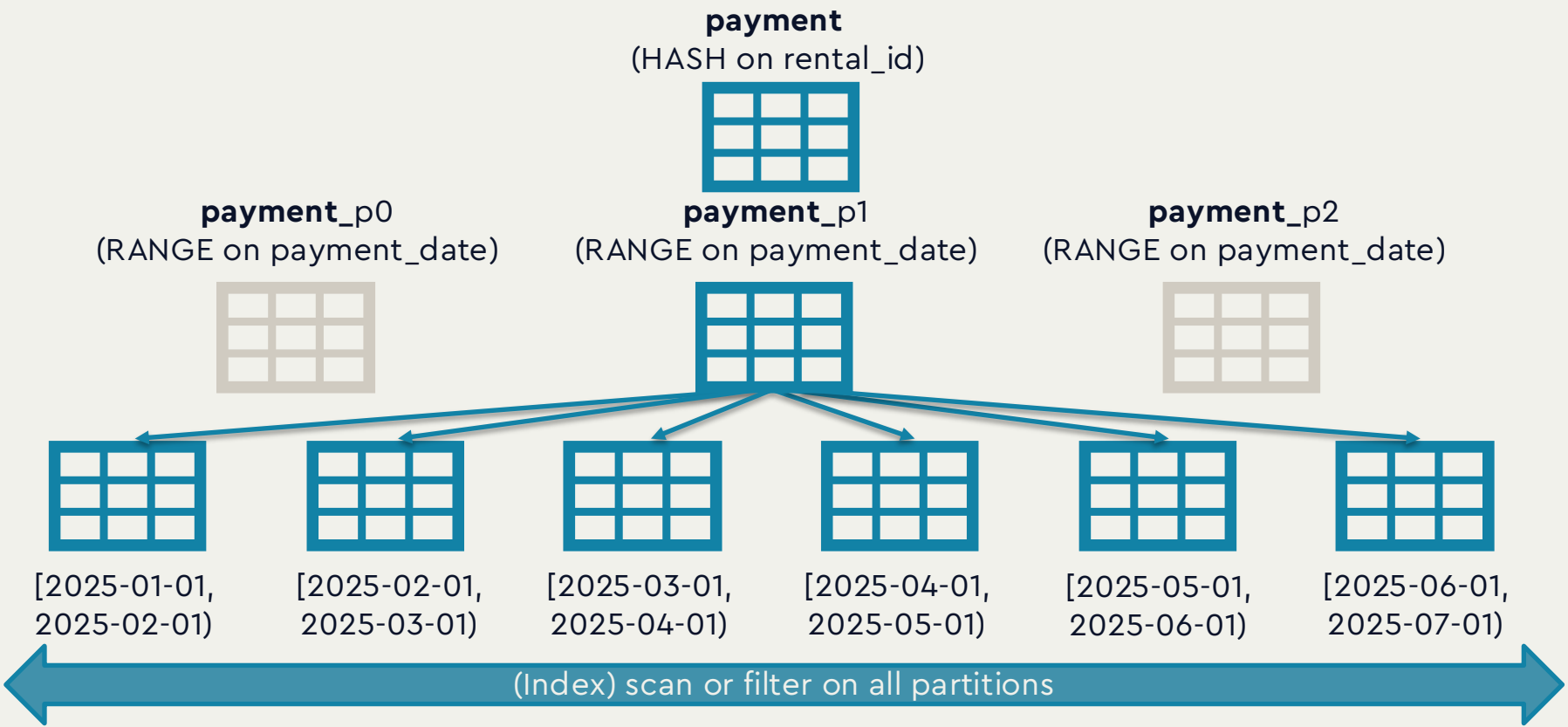
pganalyze

- Partition by primary/alternate key first (HASH or LIST)
- Sub-partition by RANGE
- More, smaller partitions and indexes
- Can increase INSERT throughput and data access concurrency*

CAUTION: Generally, increases overall maintenance for the same amount of data (more tables, indexes, etc.)

*YMMV


payment
(HASH on rental_id)
payment_p0
(RANGE on payment_date)
payment_p1
(RANGE on payment_date)
payment_p2
(RANGE on payment_date)
[2025-01-01, 2025-02-01)
[2025-02-01, 2025-03-01)
[2025-03-01, 2025-04-01)
[2025-04-01, 2025-05-01)
[2025-05-01, 2025-06-01)
[2025-06-01, 2025-07-01)




```
SELECT * FROM payment_hash p
 JOIN rental r USING (rental_id)
WHERE rental_id = 20042391;
```

```
QUERY PLAN                                                                                                                                                                             |
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
Nested Loop  (cost=4.63..567.28 rows=204 width=72) (actual time=0.285..0.355 rows=1 loops=1)                                                                                           |
  Buffers: shared hit=101                                                                                                                                                              |
  ->  Index Scan using rental_pkey on rental r  (cost=0.43..8.45 rows=1 width=49) (actual time=0.017..0.018 rows=1 loops=1)                                                            |
        Index Cond: (rental_id = 20042391)                                                                                                                                             |
        Buffers: shared hit=4                                                                                                                                                          |
  ->  Append  (cost=4.21..556.80 rows=204 width=27) (actual time=0.257..0.325 rows=1 loops=1)                                                                                          |
        Buffers: shared hit=97                                                                                                                                                         |
        ->  Bitmap Heap Scan on paymenthash_p3_202301 p_1  (cost=4.21..14.35 rows=7 width=28) (actual time=0.007..0.008 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202301_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.002..0.002 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202302 p_2  (cost=4.21..14.35 rows=7 width=28) (actual time=0.002..0.003 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202302_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.001..0.001 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202303 p_3  (cost=4.21..14.35 rows=7 width=28) (actual time=0.002..0.003 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202303_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.001..0.001 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202304 p_4  (cost=4.21..14.35 rows=7 width=28) (actual time=0.003..0.003 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202304_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.002..0.002 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202305 p_5  (cost=4.21..14.35 rows=7 width=28) (actual time=0.003..0.003 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202305_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.002..0.002 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202306 p_6  (cost=4.21..14.35 rows=7 width=28) (actual time=0.011..0.012 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
              ->  Bitmap Index Scan on paymenthash_p3_202306_rental_id_idx  (cost=0.00..4.21 rows=7 width=0) (actual time=0.001..0.001 rows=0 loops=1)                                 |
                    Index Cond: (rental_id = 20042391)                                                                                                                                 |
                    Buffers: shared hit=2                                                                                                                                              |
        ->  Bitmap Heap Scan on paymenthash_p3_202307 p_7  (cost=4.21..14.35 rows=7 width=28) (actual time=0.003..0.004 rows=0 loops=1)                                               |
              Recheck Cond: (rental_id = 20042391)                                                                                                                                     |
              Buffers: shared hit=2                                                                                                                                                    |
```

```
|
       ->  Index Scan using paymenthash_p3_202601_rental_id_idx on paymenthash_p3_202601 p_37  (cost=0.29..8.30 rows=1 width=23) (actual time=0.072..0.072 rows=0 loops=1)|
             Index Cond: (rental_id = 20042391)                                                                                                                          |
             Buffers: shared hit=2                                                                                                                                       |
           ->  Index Scan using paymenthash_p3_202602_rental_id_idx on paymenthash_p3_202602 p_38  (cost=0.29..8.30
rows=1 width=23) (actual time=0.011..0.012 rows=1 loops=1)|
                 Index Cond: (rental_id = 20042391)
|
                 Buffers: shared hit=3
|
       ->  Index Scan using paymenthash_p3_202603_rental_id_idx on paymenthash_p3_202603 p_39  (cost=0.29..8.30 rows=1 width=23) (actual time=0.007..0.007 rows=0 loops=1)|
             Index Cond: (rental_id = 20042391)                                                                                                                          |
             Buffers: shared hit=2                                                                                                                                       ||
```

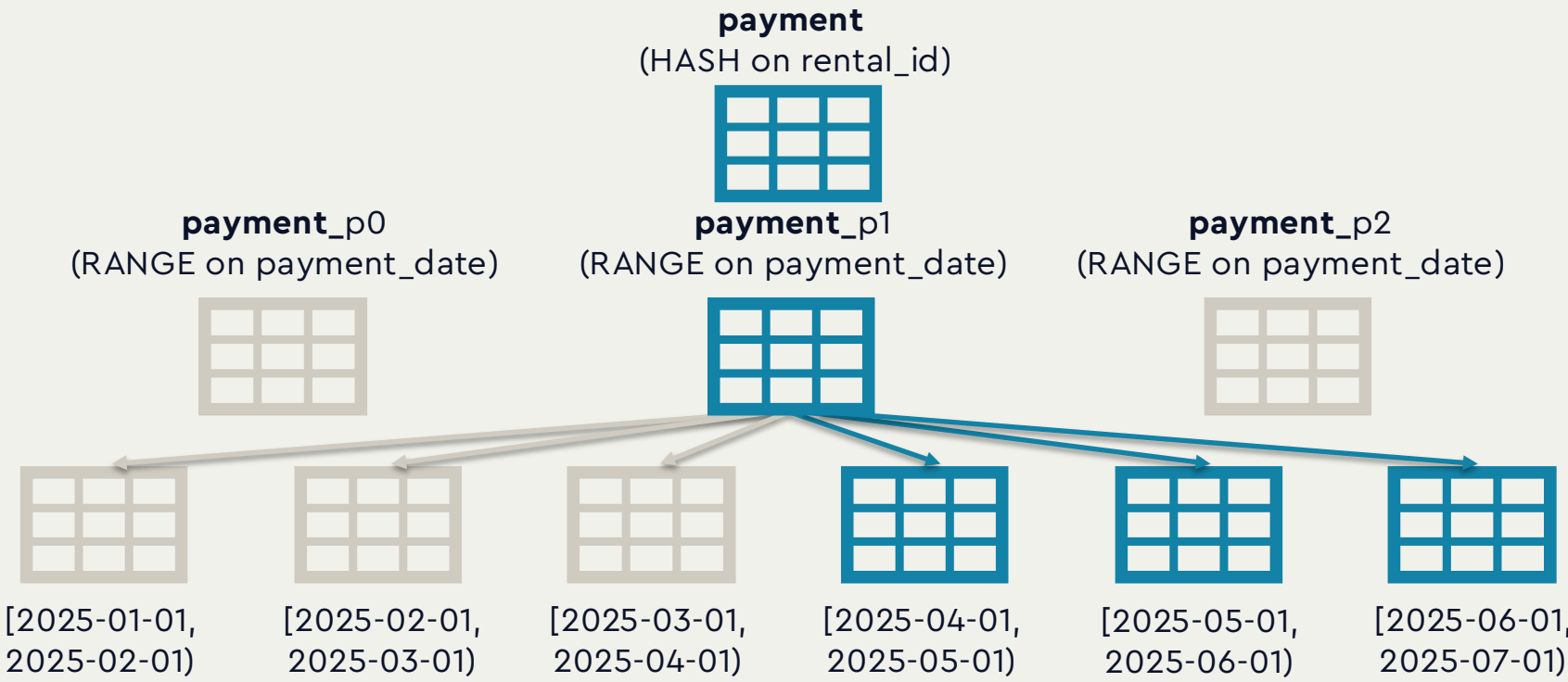
```
Planning:
  Buffers: shared hit=52
Planning Time: 1.705 ms
Execution Time: 0.610 ms
```

Runtime Partition Pruning

pganalyze

- Prune initial path by primary/alternate key value
- Use **Runtime partition pruning** to avoid data access even when a partition is originally planned
- All partitions on the second level will still be planned, but only matching partitions will be executed
- Generally* - higher planning time, lower execution time

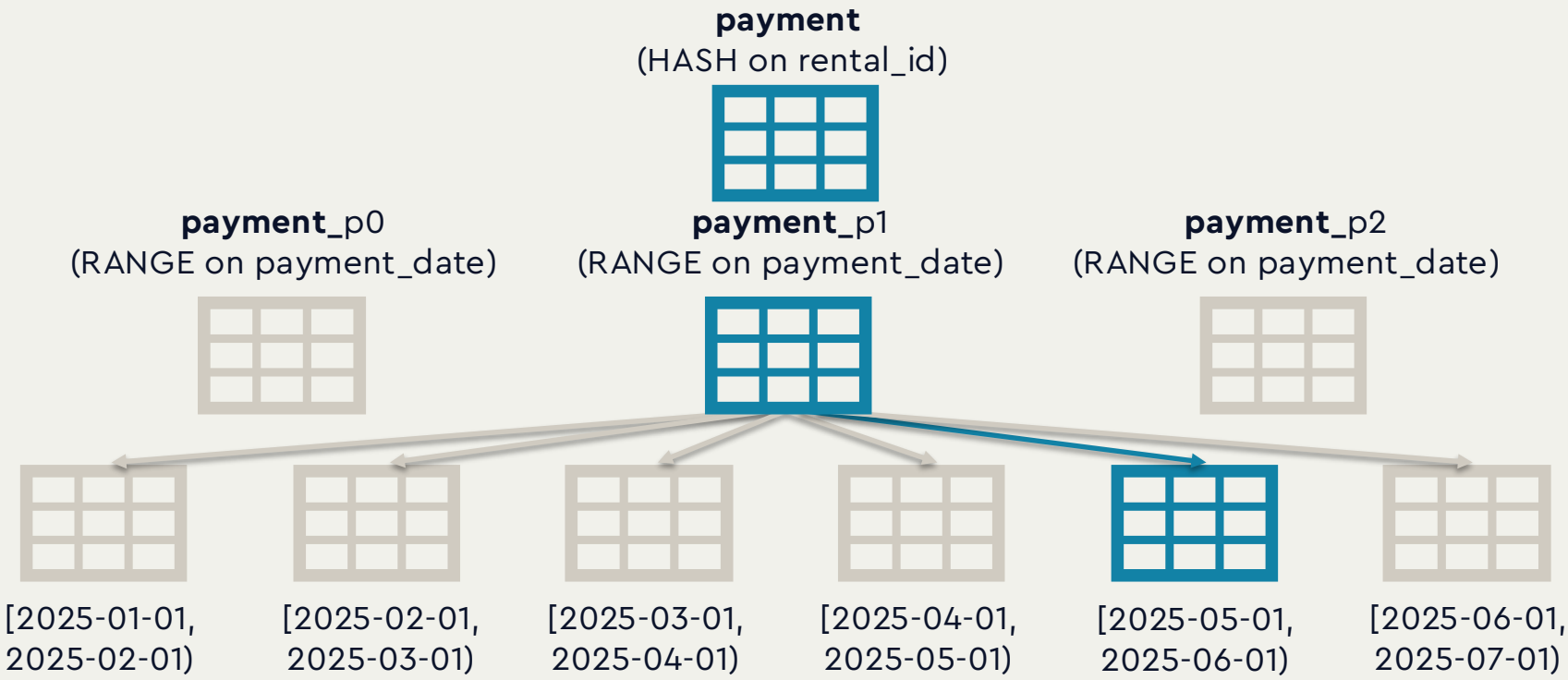
*YMMV




```
SELECT * FROM payment_hash p
JOIN rental r USING (rental_id)
WHERE rental_id = 20042391
AND payment_date >= lower(rental_period);
```

- "(never executed)" = runtime pruning
- Dynamic filter is evaluated at runtime and used to prune partitions for data access
- Reduces overall buffer access at runtime

```
->  Index Scan using paymenthash_p3_202511_rental_id_idx on paymenthash_p3_202511 p_35  (cost=0.29..8.31 rows=1 width=23) (never executed)
            Index Cond: (rental_id = 20042391)
            Filter: (payment_date >= lower(r.rental_period))
      ->  Index Scan using paymenthash_p3_202512_rental_id_idx on paymenthash_p3_202512 p_36  (cost=0.29..8.31 rows=1 width=23) (never executed)
            Index Cond: (rental_id = 20042391)
            Filter: (payment_date >= lower(r.rental_period))
      ->  Index Scan using paymenthash_p3_202601_rental_id_idx on paymenthash_p3_202601 p_37  (cost=0.29..8.31 rows=1 width=23) (never executed)
            Index Cond: (rental_id = 20042391)
            Filter: (payment_date >= lower(r.rental_period))
      ->  Index Scan using paymenthash_p3_202602_rental_id_idx on paymenthash_p3_202602 p_38  (cost=0.29..8.31 rows=1 width=23)
                                                              (actual time=0.010..0.011 rows=1 loops=1)
            Index Cond: (rental_id = 20042391)
            Filter: (payment_date >= lower(r.rental_period))
            Buffers: shared hit=3
      ->  Index Scan using paymenthash_p3_202603_rental_id_idx on paymenthash_p3_202603 p_39  (cost=0.29..8.31 rows=1 width=23)
                                                              (actual time=0.007..0.007 rows=0 loops=1)
            Index Cond: (rental_id = 20042391)
            Filter: (payment_date >= lower(r.rental_period))
            Buffers: shared hit=2
```

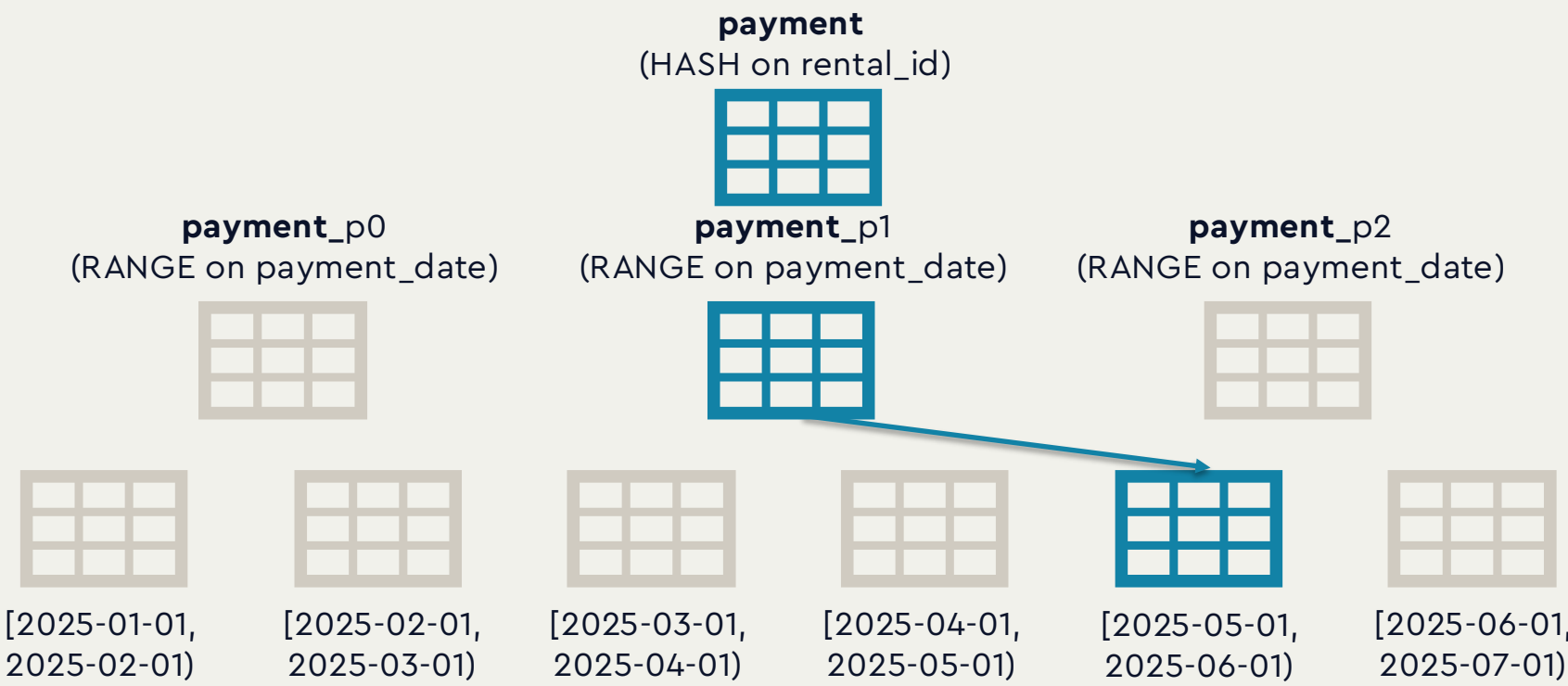



```sql
SELECT * FROM payment_hash p
JOIN rental r USING (rental_id)
WHERE rental_id = 20042391
AND payment_date >= lower(rental_period)
AND payment_date <= upper(rental_period) + '1 day'::interval;
```

Partition Pruning during planning

pganalyze

- Plan time pruning is still the most desirable
- Use static values whenever possible on partitioning keys to benefit from plan pruning
- Sub-partitions still provide:
  - Higher locality density
  - Smaller data surface area for specific keys




```sql
SELECT * FROM payment_hash p
JOIN rental r USING (rental_id)
WHERE rental_id = 20042391
AND payment_date >= '2026-02-06'
AND payment_date <= '2026-02-08';
```

```
QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
Nested Loop  (cost=0.71..16.76 rows=1 width=68) (actual time=0.120..0.122 rows=1 loops=1)
  Buffers: shared hit=7
  ->  Index Scan using paymenthash_p3_202602_rental_id_idx on paymenthash_p3_202602 p  (cost=0.29..8.31 rows=1
width=23) (actual time=0.058..0.059 rows=1 loops=1)
        Index Cond: (rental_id = 20042391)
        Filter: ((payment_date >= '2026-02-06 00:00:00-05'::timestamp with time zone) AND (payment_date <=
'2026-02-08 00:00:00-05'::timestamp with time zone))
        Buffers: shared hit=3
  ->  Index Scan using rental_pkey on rental r  (cost=0.43..8.45 rows=1 width=49)
                        (actual time=0.059..0.060 rows=1 loops=1)
        Index Cond: (rental_id = 20042391)
        Buffers: shared hit=4

Planning Time: 0.488 ms
Execution Time: 0.172 ms
```

Partition Automation

Partition Automation

pganalyze

- Don't plan on manual creation or retention
- Common extensions:
  - pg_partman/pg_cron
  - timescaledb (not declarative partitioning)
  - citus
  - EDB Postgres Distributed Server

pg_partman

pganalyze

- Automation extension for partition management
  - Creating future partitions
  - Dropping/detaching old partitions on schedule
  - "migrating" data into partitioned table
- PG14+ support w/declarative partitioning only
- Not partitioning engine
- Everything this extension does could be replicated by hand and raw SQL

```
CREATE EXTENSION pg_partman;

CREATE TABLE bluebox.payment2 (
  payment_id integer GENERATED BY DEFAULT AS IDENTITY,
  customer_id integer NOT NULL,
  rental_id integer NOT NULL,
  amount numeric(5,2) NOT NULL,
  payment_date timestamptz NOT NULL,
  CONSTRAINT payment2_pkey PRIMARY KEY (payment_date, payment_id)
)
PARTITION BY RANGE (payment_date);
```

```
-- Hand pg_partman the config
SELECT partman.create_parent(
  p_parent_table := 'bluebox.payment2’,
  p_control := 'payment_date’,
  p_interval := '1 month’,
  p_premake := 4 -- always 4 future partitions
);
```

```
UPDATE partman.part_config
SET retention = '12 months’,
 retention_keep_table = false -- drop or detach?
WHERE parent_table = 'bluebox.payment2';
```

pg_partman – Other Features

- Sub-partitioning support
- Naming and templates for configuration
  - Indexes, tablespaces, etc.
- Basic batch migration help
  - Copy from non-partitioned table into partitioned table
  - Not automated create/copy/rename/drop

Converting non-partitioned tables

"Easy" Button Approach

- Create new partitioned table
- Add necessary partitions
- COPY data from original to partitioned table

"Easy" Button Approach – Gotchas

- Double the data for some period of time
- Longer periods of potential locking/blocking

Alternate "partition forward" approach

- Create partitioned table
- Add new partitions
- Create CHECK constraints on existing, non-partitioned table
- Rename & attach existing table as partition
- Rename new partitioned table

```sql
CREATE TABLE bluebox.payment_new (
   LIKE bluebox.payment INCLUDING ALL
) PARTITION BY RANGE (payment_date);

-- be careful here - there can't be any overlap in partition values
CREATE TABLE bluebox.payment_new_2026_05 PARTITION OF bluebox.payment_new
FOR VALUES FROM ('2026-05-01') TO ('2026-06-01');


CREATE TABLE bluebox.payment_new_2026_06 PARTITION OF bluebox.payment_new
FOR VALUES FROM ('2026-06-01') TO ('2026-07-01');
```

```sql
-- NOT VALID takes a very short exclusive lock to record
-- the constraint only
ALTER TABLE bluebox.payment
  ADD CONSTRAINT payment_historical_bound
  CHECK (payment_date < '2026-05-01') NOT VALID;


-- VALIDATE just takes a shared lock in this case
-- READ/WRITE can still take place
ALTER TABLE bluebox.payment VALIDATE CONSTRAINT payment_historical_bound;
```

```
-- Do the rename in one transaction!
BEGIN;


-- rename the old table
ALTER TABLE bluebox.payment RENAME TO payment_historical;


-- replace MINVALUE with a reasonable value before start of data
ALTER TABLE bluebox.payment_new
  ATTACH PARTITION bluebox.payment_historical
  FOR VALUES FROM (MINVALUE) TO ('2026-05-01');

-- Now rename the partitioned table
ALTER TABLE bluebox.payment_new RENAME TO payment;


COMMIT;
```

Managing the Default Partition

Default Partition

- Special "catch all" partition
- Prevents data loss
- Maintenance headache as new partitions are created
- TL;DR; - often a necessary ~~evil~~ help

```
CREATE TABLE bluebox.payment2_default
       PARTITION OF bluebox.payment2 DEFAULT;

CREATE TABLE bluebox.payment2_y2024m01
       PARTITION OF bluebox.payment2
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');

CREATE TABLE bluebox.payment2_y2024m02
       PARTITION OF bluebox.payment2
FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');

CREATE TABLE bluebox.payment2_y2024m03
       PARTITION OF bluebox.payment2
FOR VALUES FROM ('2024-03-01') TO ('2024-04-01');
```

Moving data to partitions

- Similar process to converting an existing non-partitioned table into partition
  - Create new table LIKE parent
  - INSERT data from default partition into new table
  - Add CHECK constraint and validate
  - DELETE data from default partition
  - ATTACH new partition

Merging/Splitting existing partitions

Why Merge or Split partitions

pganalyze

- Chose too narrow or broad partition length
- Reduce partitions over time
  - Weekly → monthly
  - Monthly → quarterly
- Fewer partitions to plan
- RANGE and LIST types only

Merging or Splitting Partitions

- Not available in PG <= 18
- Currently committed in PG19-dev!
- Alternatively, do manually given previous discussions on attaching/detaching partitions
    - Remember to create appropriate constraints ahead of time

```
CREATE TABLE bluebox.payment_merge_demo (
  payment_id integer GENERATED BY DEFAULT AS IDENTITY,
  customer_id integer NOT NULL,
  rental_id integer NOT NULL,
  amount numeric(5,2) NOT NULL,
  payment_date timestamptz NOT NULL,
  CONSTRAINT payment_merge_demo_pkey PRIMARY KEY (payment_date, payment_id)
) PARTITION BY RANGE (payment_date);
```

```
CREATE TABLE bluebox.payment_merge_demo_2026_01 PARTITION OF
   bluebox.payment_merge_demo
FOR VALUES FROM ('2026-01-01') TO ('2026-02-01');

CREATE TABLE bluebox.payment_merge_demo_2026_02 PARTITION OF
   bluebox.payment_merge_demo
FOR VALUES FROM ('2026-02-01') TO ('2026-03-01');

CREATE TABLE bluebox.payment_merge_demo_2026_03 PARTITION OF
   bluebox.payment_merge_demo
FOR VALUES FROM ('2026-03-01') TO ('2026-04-01');

-- etc.
```

```
partition                   |count|
----------------------------+-----+
payment_merge_demo_2026_04|65869|
payment_merge_demo_2026_01|96461|
payment_merge_demo_2026_02|83120|
payment_merge_demo_2026_03|83443|
```

```
ALTER TABLE bluebox.payment_merge_demo
MERGE PARTITIONS (
  bluebox.payment_merge_demo_2026_01,
  bluebox.payment_merge_demo_2026_02,
  bluebox.payment_merge_demo_2026_03
)
INTO bluebox.payment_merge_demo_2026_q1;
```

```
partition                    |count |
-----------------------------+------+
payment_merge_demo_2026_04| 65869|
payment_merge_demo_2026_q1|263024|
```

```
ALTER TABLE bluebox.payment_merge_demo
SPLIT PARTITION bluebox.payment_merge_demo_2026_q1
INTO (
  PARTITION bluebox.payment_merge_demo_2026_01
    FOR VALUES FROM ('2026-01-01') TO ('2026-02-01'),
  PARTITION bluebox.payment_merge_demo_2026_02
    FOR VALUES FROM ('2026-02-01') TO ('2026-03-01'),
  PARTITION bluebox.payment_merge_demo_2026_03
    FOR VALUES FROM ('2026-03-01') TO ('2026-04-01')
);
```

```
partition                    |count|
-----------------------------+-----+
payment_merge_demo_2026_04|65869|
payment_merge_demo_2026_01|96461|
payment_merge_demo_2026_02|83120|
payment_merge_demo_2026_03|83443|
```

Partition Size and Number

Partition size and number

- shared_buffers 25%+/- memory
- Hot partitions should fit in shared buffers
- For large partitions, ensure relevant indexes are available after partition exclusion
- Don't try to target row count as your target
- Consider data retention

Partition size and number

pganalyze

- Too many partitions can increase planning time
  - >~1,000 partitions may require constraint/range changes
  - <=PG13 may struggle with LWLocks
- Locks:
  - PG14+ all planned objects receive initial lock
  - Many partitions with many indexes (for instance) can explode the available number of LWLocks
  - Queries end up waiting lock management
  - https://www.kylehailey.com/post/postgres-partition-pains-lockmanager-waits

Finally...

Revisit earlier discussion about sub-partitions and partition pruning for better plan hygiene

Data Tiering

Use Schemas and Tablespace

pganalyze

- Consider creating partitions in a separate schema
- Separate tablespaces = data tiering
  - Hot data in fastest memory-based storage
  - Warm data in cheaper storage
  - Cold data to disk or object storage

Data Tiering

```
-- Partition to a new tablespace
ALTER TABLE payment2_y2024m02 SET TABLESPACE slow_ts;

-- Partition index to a new tablespace
ALTER INDEX rental2_y2023m01_rental_period_idx
      SET TABLESPACE slow_ts;
```

Recent Updates and Improvements

Postgres 15

pganalyze

- **CLUSTER on partitioned tables** (no more per-partition loop)
- **Row-movement UPDATEs with FKs** run as UPDATE on partition root
- **Planning-time** improvements for queries on partitioned tables

Postgres 17

pganalyze

- **Identity columns** on partitioned tables work as expected in all cases
- **Exclusion constraints** on partitioned tables

Postgres 18

pganalyze

- Planning **efficiency** for queries accessing many partitions
- **Partitionwise joins**: more cases handled, less planner memory
    - makes **`enable_partitionwise_join=on`** viable in production
- Improved **cost estimates** for partition queries
- **NOT VALID** foreign keys on partitioned tables

Postgres 18

pganalyze

- **`ALTER TABLE ... DROP CONSTRAINT ... ONLY`** on partitioned tables now works
- **VACUUM/ANALYZE ONLY**
  - process the partitioned parent without cascading into children
- Unlogged partitioned tables disallowed
- **Fast-path lock** slots are finally tunable via **`max_locks_per_transaction`**
  - first real DBA knob for the lock-manager contention problem

What Questions do you have?

Thank you!

Learn more at pganalyze.com

Contact us: ryan@pganalyze.com