Детален опис на партиционирањето

Партиционирањето е техника каде што една голема табела се дели на помали физички делови, наречени партиции. Во нашиот случај, табелата Ticket, која содржи 15 милиони редови, се дели по години врз основа на колоната purchase_date. Тоа значи дека сите билети купени во 1999 година се складираат во посебна партиција наречена ticket_1999, сите билети од 2000 година во ticket_2000, и така натаму до 2027 година.

```sql
CREATE TABLE Ticket_Partitioned (
    ticket_id SERIAL,
    showtime_id INT NOT NULL,
    seat_id INT NOT NULL,
    reservation_id INT NOT NULL,
    price INT NOT NULL,
    purchase_date DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (ticket_id, purchase_date),
    FOREIGN KEY (showtime_id) REFERENCES Showtime(showtime_id),
    FOREIGN KEY (seat_id) REFERENCES Seat(seat_id),
    FOREIGN KEY (reservation_id) REFERENCES Reservation(reservation_id)
) PARTITION BY RANGE (purchase_date);
```

```sql
CREATE TABLE ticket_1999 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('1999-01-01') TO ('2000-01-01');
CREATE TABLE ticket_2000 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2000-01-01') TO ('2001-01-01');
CREATE TABLE ticket_2001 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2001-01-01') TO ('2002-01-01');
CREATE TABLE ticket_2002 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2002-01-01') TO ('2003-01-01');
CREATE TABLE ticket_2003 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2003-01-01') TO ('2004-01-01');
CREATE TABLE ticket_2004 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2004-01-01') TO ('2005-01-01');
CREATE TABLE ticket_2005 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2005-01-01') TO ('2006-01-01');
CREATE TABLE ticket_2006 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2006-01-01') TO ('2007-01-01');
CREATE TABLE ticket_2007 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2007-01-01') TO ('2008-01-01');
CREATE TABLE ticket_2008 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2008-01-01') TO ('2009-01-01');
CREATE TABLE ticket_2009 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2009-01-01') TO ('2010-01-01');
CREATE TABLE ticket_2010 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2010-01-01') TO ('2011-01-01');
CREATE TABLE ticket_2011 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2011-01-01') TO ('2012-01-01');
CREATE TABLE ticket_2012 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2012-01-01') TO ('2013-01-01');
CREATE TABLE ticket_2013 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2013-01-01') TO ('2014-01-01');
CREATE TABLE ticket_2014 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2014-01-01') TO ('2015-01-01');
CREATE TABLE ticket_2015 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2015-01-01') TO ('2016-01-01');
CREATE TABLE ticket_2016 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2016-01-01') TO ('2017-01-01');
CREATE TABLE ticket_2017 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2017-01-01') TO ('2018-01-01');
CREATE TABLE ticket_2018 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2018-01-01') TO ('2019-01-01');
CREATE TABLE ticket_2019 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2019-01-01') TO ('2020-01-01');
CREATE TABLE ticket_2020 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2020-01-01') TO ('2021-01-01');
CREATE TABLE ticket_2021 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2021-01-01') TO ('2022-01-01');
CREATE TABLE ticket_2022 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2022-01-01') TO ('2023-01-01');
CREATE TABLE ticket_2023 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');
CREATE TABLE ticket_2024 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');
CREATE TABLE ticket_2025 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');
CREATE TABLE ticket_2026 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2026-01-01') TO ('2027-01-01');
CREATE TABLE ticket_2027 PARTITION OF Ticket_Partitioned FOR VALUES FROM ('2027-01-01') TO ('2028-01-01');
```

Предностите на партиционирањето се видливи во практичните резултати. Прво, пребарувањата стануваат значително побрзи. Кога ќе се постави услов што филтрира по година, PostgreSQL автоматски ги прескокнува останатите партиции и ја пребарува само потребната. Ова се нарекува partition pruning. На пример, при пребарување на сите билети од 2020 година каде што се брои вкупниот број на редови, времето на извршување без партиционирање беше 852 милисекунди, додека со партиционирање се намали на само 52 милисекунди. Тоа значи дека партиционирањето го забрза пребарувањето 16 пати.

```
[2026-05-25 16:20:59] NAJJAKA.public> SELECT COUNT(*) FROM Ticket
                                      WHERE purchase_date >= '2020-01-01' AND purchase_date < '2021-01-01'
[2026-05-25 16:21:00] 1 row retrieved starting from 1 in 1 s 179 ms (execution: 852 ms, fetching: 327 ms)
```

```
[2026-05-25 16:50:58] NAJJAKA.public> SELECT COUNT(*) FROM Ticket
                                      WHERE purchase_date >= '2020-01-01' AND purchase_date < '2021-01-01'
[2026-05-25 16:50:58] 1 row retrieved starting from 1 in 370 ms (execution: 52 ms, fetching: 318 ms)
```

Вториот практичен резултат е уште повпечатлив – при комплексен JOIN со четири табели (Ticket, Reservation, Showtime, Movie) кој филтрира по година и враќа 500 редови, времето на извршување се намали од 1377 милисекунди на 707 милисекунди, односно скоро 2 пати побрзо. Второ, управувањето на табелата е полесно бидејќи може да се избрише или архивира цела година без да се допираат останатите податоци. Трето, индексите работат поефикасно затоа што секоја партиција содржи помал број редови.

```
[2026-05-25 16:22:00] NAJJAKA.public> SELECT t.ticket_id, t.price, m.title, r.status
                                      FROM Ticket t
                                      JOIN Reservation r ON r.reservation_id = t.reservation_id
                                      JOIN Showtime s ON s.showtime_id = t.showtime_id
                                      JOIN Movie m ON m.movie_id = s.movie_id
                                      WHERE t.purchase_date >= '2020-01-01' AND t.purchase_date < '2021-01-01'
[2026-05-25 16:22:02] 500 rows retrieved starting from 1 in 2 s 255 ms (execution: 1 s 922 ms, fetching: 333 ms)
```

```
[2026-05-25 16:52:01] NAJJAKA.public> SELECT t.ticket_id, t.price, m.title, r.status
                                     FROM Ticket t
                                     JOIN Reservation r ON r.reservation_id = t.reservation_id
                                     JOIN Showtime s ON s.showtime_id = t.showtime_id
                                     JOIN Movie m ON m.movie_id = s.movie_id
                                     WHERE t.purchase_date >= '2020-01-01' AND t.purchase_date < '2021-01-01'
[2026-05-25 16:52:02] 500 rows retrieved starting from 1 in 1 s 707 ms (execution: 1 s 377 ms, fetching: 330 ms)
```

Во PostgreSQL, RANGE партиционирањето значи дека секоја партиција покрива одреден опсег на вредности. На пример, партицијата ticket_2000 ги содржи сите датуми од 1 јануари 2000 година до 31 декември 2000 година. Кога ќе се внесе нов билет со purchase_date од 15 март 2000 година, PostgreSQL автоматски го сместува во соодветната партиција ticket_2000.

Primary key треба да ја содржи колоната според која се врши партиционирањето (purchase_date), бидејќи PostgreSQL мора да знае во која партиција ќе биде сместен секој ред. Доколку primary key би содржел само ticket_id, PostgreSQL не би можел ефикасно да управува со партициите.

Индексите кај партиционираните табели функционираат слично – индексот idx_ticket_showtime_id се создава посебно за секоја партиција. Ова го прави индексирањето поефикасно бидејќи секој индекс работи со помал број редови. При пребарување кое филтрира по година, PostgreSQL го користи partition pruning за да прескокне непотребни индекси и брзо да пристапи до соодветната партиција.

Кога се внесуваат 15 милиони редови во партиционирана табела, PostgreSQL автоматски ја распределува секоја редица во соодветната партиција врз основа на нејзиниот purchase_date. Иако процесот на внесување е малку посложен отколку кај обична табела, добивката е значително подобра брзина при пребарување и анализа на податоците. Вистинските резултати покажуваат дека разликата е значајна, особено при пребарувања што филтрираат по датум, каде што брзината се подобрува и до 16 пати, што го прави партиционирањето многу корисна оптимизација за големи табели.