Напредна тема – Партиционирање

Партиционирањето претставува техника за поделба на големи табели на повеќе помали физички делови (партиции), при што за корисникот тие и понатаму се однесуваат како една табела. Во нашиот систем за продажба на билети оваа техника е применета со цел да се подобрат перформансите на пребарувањата, да се намали времето на извршување на комплексни барања и да се овозможи поефикасно управување со големи количини податоци. Со користење на различни стратегии за партиционирање (RANGE, LIST и HASH) податоците се организираат според нивните карактеристики, што овозможува базата на податоци да обработува само релевантен дел од податоците наместо целата табела.

1. RANGE Partitioning на Ticket според purchased_at

Кај табелата Ticket е имплементирано RANGE партиционирање врз основа на атрибутот purchased_at. Податоците се распределуваат во квартални партиции според датумот на купување на билетот. На овој начин, кога системот извршува пребарување за одреден временски период, базата пристапува само до партицијата која ги содржи потребните записи. Ова значително го намалува бројот на прочитани редови и ги подобрува перформансите на аналитичките и финансиските извештаи кои најчесто се филтрираат според датум на купување.

Податоците автоматски се насочуваат кон соодветната физичка партиција во зависност од датумот на купување на билетот. На пример, доколку се изврши финансиско пребарување со услов WHERE purchased_at BETWEEN '2026-01-15' AND '2026-02-15', оптимизаторот на барања (query planner) ќе ги игнорира сите останати партиции и ќе пристапи само до партицијата ticket_range_y2026_q1. На овој начин се намалува времето потребно за извршување на барањето и се подобрува ефикасноста на системот.

```
CREATE TABLE ticket_range (
    ticket_id BIGINT GENERATED ALWAYS AS IDENTITY,
    status VARCHAR(50) NOT NULL DEFAULT 'AVAILABLE',
    reserved_at TIMESTAMP,
    expires_at TIMESTAMP,
    purchased_at TIMESTAMP NOT NULL,
    ticket_price NUMERIC(10,2) NOT NULL,
    qr_code VARCHAR(255) NOT NULL,
    customer_id BIGINT,
    seat_id BIGINT NOT NULL,
    event_id BIGINT NOT NULL,
    ticket_type_id BIGINT NOT NULL,
    payment_id BIGINT,

    PRIMARY KEY (ticket_id, purchased_at)
) PARTITION BY RANGE (purchased_at);


CREATE TABLE ticket_range_y2026_q1 PARTITION OF ticket_range
    FOR VALUES FROM ('2026-01-01 00:00:00') TO ('2026-04-01 00:00:00');

CREATE TABLE ticket_range_y2026_q2 PARTITION OF ticket_range
    FOR VALUES FROM ('2026-04-01 00:00:00') TO ('2026-07-01 00:00:00');

CREATE TABLE ticket_range_default PARTITION OF ticket_range DEFAULT;
```

2. LIST Partitioning на Ticket според status

Кај табелата Ticket е применето LIST партиционирање според атрибутот status. Активните билети (AVAILABLE, RESERVED, PURCHASED и SCANNED) се чуваат во една партиција, додека откажаните билети (CANCELLED) се чуваат во посебна партиција. Ова овозможува оперативните процеси на системот, како што се резервација, купување и валидација на билети, да работат само со активните записи без да бидат оптоварени од историските или неактивните податоци. На тој начин се подобрува брзината на секојдневните операции и се намалува времето потребно за извршување на барањата.

Со оваа поделба активните билети се одделуваат од неактивните записи. На пример, доколку скенерите на влезот на настанот вршат проверка на билетите со услов WHERE status = 'PURCHASED', системот ќе пристапи само до партицијата ticket_active. Историјата на откажаните билети ќе биде целосно изолирана во посебна партиција, со што нема да влијае врз перформансите на операциите кои се извршуваат во реално време.

```
CREATE TABLE ticket_list (
    ticket_id BIGINT GENERATED ALWAYS AS IDENTITY,
    status VARCHAR(50) NOT NULL DEFAULT 'AVAILABLE',
    reserved_at TIMESTAMP,
    expires_at TIMESTAMP,
    purchased_at TIMESTAMP,
    ticket price NUMERIC(10,2) NOT NULL,
    qr_code VARCHAR(255) NOT NULL,
    customer_id BIGINT,
    seat_id BIGINT NOT NULL,
    event_id BIGINT NOT NULL,
    ticket_type_id BIGINT NOT NULL,
    payment_id BIGINT,
    PRIMARY KEY (ticket_id, status)
) PARTITION BY LIST (status);

CREATE TABLE ticket_active PARTITION OF ticket_list
    FOR VALUES IN ('AVAILABLE', 'RESERVED', 'PURCHASED', 'SCANNED');

CREATE TABLE ticket_inactive PARTITION OF ticket_list
    FOR VALUES IN ('CANCELLED');
```

3. HASH Partitioning на Payment според customer_id

Кај табелата Payment е имплементирано HASH партиционирање според атрибутот customer_id. Овој пристап обезбедува рамномерна распределба на податоците помеѓу повеќе партиции, без оглед на бројот на плаќања што ги извршува секој корисник. Бидејќи системот обработува голем број трансакции, HASH партиционирањето спречува создавање на преголеми партиции и овозможува подобро распределување на оптоварувањето. Како резултат на тоа, пребарувањата и обработката на плаќањата се извршуваат поефикасно, а системот полесно се справува со зголемен број корисници и трансакции.

Партиционирањето на табелата Payment преку HASH функција над атрибутот customer_id обезбедува рамномерна распределба на записите низ сите партиции. Бидејќи еден корисник може да изврши повеќе плаќања со текот на времето, ваквиот пристап спречува појава на нерамномерна распределба на податоците (data skew) и овозможува подобро балансирање на оптоварувањето. За разлика од RANGE партиционирањето, HASH партиционирањето не се базира на логички редослед на податоците, туку на нивна рамномерна распределба, што го прави особено соодветно за системи со голем број трансакции и интензивна обработка на податоци.

```
CREATE TABLE payment_hash (
    payment_id BIGINT GENERATED ALWAYS AS IDENTITY,
    amount NUMERIC(10,2) NOT NULL CHECK (amount >= 0),
    payment_method VARCHAR(100) NOT NULL,
    payment_status VARCHAR(50) NOT NULL,
    payment_date TIMESTAMP NOT NULL,
    customer_id BIGINT NOT NULL,
    discount_id BIGINT,

    PRIMARY KEY (payment_id, customer_id),

    CHECK (payment_status IN ('PENDING','COMPLETED','FAILED','REFUNDED')),
    CHECK (payment_method IN ('CASH','CARD','ONLINE','TRANSFER')),
    CHECK (payment_date <= CURRENT_TIMESTAMP)
)
PARTITION BY HASH (customer_id);
CREATE TABLE payment_p0 PARTITION OF payment_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 0);

CREATE TABLE payment_p1 PARTITION OF payment_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 1);

CREATE TABLE payment_p2 PARTITION OF payment_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 2);

CREATE TABLE payment_p3 PARTITION OF payment_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```