Mastering Advanced SQL for SaaS Analytics: Cohort Retention, Revenue Attribution & Performance Optimization at Scale

/*

Objective:

1. Analyze user engagement on a multi-tenant SaaS platform.

2. Compute monthly active users (MAU), retention cohorts, and revenue attribution.

3. Optimize query performance for large-scale data warehouses.

4. Use advanced SQL features: window functions, CTEs, lateral joins, JSON handling, materialized views, and query hints.

Keywords: SQL optimization, window functions, CTE, lateral join, JSONB, partitioning, indexing, materialized views, cohort analysis, revenue attribution, SaaS analytics.

*/

/* Step 1: Prepare data with Common Table Expressions (CTEs) for modularity */

WITH

-- Extract events related to user logins and purchases for performance

raw_events AS (

SELECT

tenant_id,

user_id,

```
        event_type,
        event_timestamp,
        event_metadata::jsonb,  -- JSONB column for flexible schema
        DATE_TRUNC('month', event_timestamp) AS event_month
    FROM
        events
    WHERE
        event_type IN ('login', 'purchase')
        AND event_timestamp >= CURRENT_DATE - INTERVAL '18 months'
),

-- Identify user first login month per tenant for cohort grouping
user_cohorts AS (
    SELECT
        tenant_id,
        user_id,
        MIN(DATE_TRUNC('month', event_timestamp)) AS cohort_month
    FROM
        raw_events
    WHERE
        event_type = 'login'
    GROUP BY
        tenant_id,
        user_id
),
```

```
-- Monthly Active Users (MAU) by tenant and month
monthly_active_users AS (
    SELECT
        tenant_id,
        event_month,
        COUNT(DISTINCT user_id) AS mau
    FROM
        raw_events
    WHERE
        event_type = 'login'
    GROUP BY
        tenant_id,
        event_month
),

-- Cohort retention matrix: calculate how many users from a cohort were active in subsequent months
cohort_retention AS (
    SELECT
        uc.tenant_id,
        uc.cohort_month,
        re.event_month,
        COUNT(DISTINCT uc.user_id) AS active_users
    FROM
```

```
        user_cohorts uc
    INNER JOIN
        raw_events re ON uc.user_id = re.user_id
        AND uc.tenant_id = re.tenant_id
        AND re.event_type = 'login'
        AND re.event_month >= uc.cohort_month
    GROUP BY
        uc.tenant_id,
        uc.cohort_month,
        re.event_month
),

-- Calculate retention rate relative to cohort size
cohort_sizes AS (
    SELECT
        tenant_id,
        cohort_month,
        COUNT(DISTINCT user_id) AS cohort_size
    FROM
        user_cohorts
    GROUP BY
        tenant_id,
        cohort_month
),
```

```
cohort_retention_rates AS (
    SELECT
        cr.tenant_id,
        cr.cohort_month,
        cr.event_month,
        cr.active_users,
        cs.cohort_size,
        ROUND((cr.active_users::DECIMAL / cs.cohort_size) * 100, 2) AS retention_percentage
    FROM
        cohort_retention cr
    JOIN
        cohort_sizes cs ON cr.tenant_id = cs.tenant_id AND cr.cohort_month = cs.cohort_month
),

-- Revenue attribution by month and tenant with LATERAL join to extract nested JSON purchase data
monthly_revenue AS (
    SELECT
        tenant_id,
        event_month,
        SUM((purchase_data->>'amount')::NUMERIC) AS total_revenue,
        COUNT(*) AS purchase_count
    FROM
        raw_events,
        LATERAL (
```

```
            SELECT event_metadata->'purchase' AS purchase_data
        ) AS pd
    WHERE
        event_type = 'purchase'
        AND purchase_data IS NOT NULL
    GROUP BY
        tenant_id,
        event_month
),

-- Rank top 10 tenants by revenue for reporting and indexing optimization
top_tenants AS (
    SELECT
        tenant_id,
        SUM(total_revenue) OVER () AS total_revenue_all,
        SUM(total_revenue) AS tenant_revenue,
        RANK() OVER (ORDER BY SUM(total_revenue) DESC) AS revenue_rank
    FROM
        monthly_revenue
    GROUP BY
        tenant_id
    HAVING
        SUM(total_revenue) > 0
),
```

```
-- Materialized view creation simulated via CTE for performance (create actual materialized view in production)
-- This aggregates tenant-wise monthly revenue & retention for BI dashboards
tenant_monthly_summary AS (
    SELECT
        mau.tenant_id,
        mau.event_month,
        mau.mau,
        COALESCE(mr.total_revenue, 0) AS total_revenue,
        COALESCE(crr.retention_percentage, 0) AS retention_percentage
    FROM
        monthly_active_users mau
    LEFT JOIN
        monthly_revenue mr ON mau.tenant_id = mr.tenant_id AND mau.event_month = mr.event_month
    LEFT JOIN
        cohort_retention_rates crr ON mau.tenant_id = crr.tenant_id AND mau.event_month = crr.event_month
    WHERE
        mau.event_month >= CURRENT_DATE - INTERVAL '12 months'
)

/* Final Select: Show top tenants monthly summary with optimization hints */
SELECT
    tms.tenant_id,
    tms.event_month,
```

```
    tms.mau,
    tms.total_revenue,
    tms.retention_percentage,
    tt.revenue_rank
FROM
    tenant_monthly_summary tms
JOIN
    top_tenants tt ON tms.tenant_id = tt.tenant_id
WHERE
    tt.revenue_rank <= 10
ORDER BY
    tt.revenue_rank ASC,
    tms.event_month DESC
-- Query hints (example for PostgreSQL) to influence planner:
-- /*+ IndexScan(events_idx_event_type_timestamp) */
LIMIT 100;
```

Explanation & Highlights:

- **Modular Design with CTEs:** Using multiple CTEs to break down the problem step-by-step: raw event extraction, cohort assignment, MAU calculation, retention matrix, revenue attribution, and ranking.
- **Window Functions:** Used RANK() and aggregate window functions for ranking tenants by revenue.
- **LATERAL JOIN:** Extract nested JSON purchase data inline, demonstrating advanced use of lateral joins for semi-structured data.
- **JSONB Handling:** Shows how to extract and cast JSON fields (event_metadata::jsonb, purchase_data->>'amount').

- **Cohort Analysis:** Classic SaaS metric computed with first login cohort + monthly retention rates.
- **Revenue Attribution:** Aggregates revenue per tenant per month from event logs.
- **Performance Considerations:**
    - Filtered events on relevant event types.
    - Usage of indexes recommended in comments.
    - Potential materialized view for tenant monthly summary to speed up dashboards.
    - Limited result set to top tenants.
- **SaaS Multi-Tenant Focus:** Tenant scoping in every query to handle multi-tenant environments — a common real-world challenge.
- **SEO & Technical Writing Keywords Embedded:** Through comments and variable naming, aligning with the keywords like “SQL optimization,” “window functions,” “JSONB,” “cohort analysis,” and “revenue attribution.”