Mastering Advanced SQL Techniques for Scalable Data Analytics and Performance Optimization

Objective: Showcase expertise in complex SQL, data modeling, optimization, and analytics

Keywords: Window Functions, Recursive CTEs, Indexing, Partitioning, JSON Handling, Performance Tuning

Use Case: E-Commerce Analytics - Customer Segmentation, Product Insights, Revenue Forecasting

**********/

```
-- 1. Recursive CTE to generate a hierarchical category tree
WITH RECURSIVE category_hierarchy AS (
    SELECT
        category_id,
        category_name,
        parent_id,
        1 AS level
    FROM categories
    WHERE parent_id IS NULL
    UNION ALL
    SELECT
        c.category_id,
        c.category_name,
```

```
        c.parent_id,
        ch.level + 1
    FROM categories c
    INNER JOIN category_hierarchy ch
        ON c.parent_id = ch.category_id
)
SELECT *
FROM category_hierarchy
ORDER BY level, parent_id, category_id;


-- 2. Complex Window Functions for customer lifetime value (LTV) and retention analysis
WITH customer_orders AS (
    SELECT
        o.customer_id,
        o.order_id,
        o.order_date,
        o.total_amount,
        RANK() OVER (PARTITION BY o.customer_id ORDER BY o.order_date ASC) AS first_order_rank,
        ROW_NUMBER() OVER (PARTITION BY o.customer_id ORDER BY o.order_date DESC) AS last_order_rank,
        SUM(o.total_amount) OVER (PARTITION BY o.customer_id ORDER BY o.order_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_spent,
        COUNT(o.order_id) OVER (PARTITION BY o.customer_id) AS total_orders
    FROM orders o
)
```

```
SELECT
    co.customer_id,
    co.total_orders,
    co.cumulative_spent AS lifetime_value,
    MIN(co.order_date) AS first_order_date,
    MAX(co.order_date) AS last_order_date,
    DATEDIFF(DAY, MIN(co.order_date), MAX(co.order_date)) AS customer_lifespan_days
FROM customer_orders co
GROUP BY co.customer_id, co.total_orders, co.cumulative_spent
HAVING co.cumulative_spent > 500
ORDER BY lifetime_value DESC;

-- 3. JSON handling for dynamic product attributes
SELECT
    p.product_id,
    p.product_name,
    p.category_id,
    JSON_VALUE(p.product_attributes, '$.color') AS color,
    JSON_VALUE(p.product_attributes, '$.size') AS size,
    JSON_QUERY(p.product_attributes, '$.tags') AS tags_array
FROM products p
WHERE JSON_VALUE(p.product_attributes, '$.warranty') = '2 years';

-- 4. Advanced aggregation with GROUPING SETS, ROLLUP, and CUBE
SELECT
```

```
    c.category_name,
    EXTRACT(YEAR FROM o.order_date) AS order_year,
    EXTRACT(MONTH FROM o.order_date) AS order_month,
    SUM(o.total_amount) AS total_revenue,
    COUNT(DISTINCT o.customer_id) AS unique_customers
FROM orders o
INNER JOIN products p ON o.product_id = p.product_id
INNER JOIN categories c ON p.category_id = c.category_id
GROUP BY CUBE (c.category_name, EXTRACT(YEAR FROM o.order_date),
EXTRACT(MONTH FROM o.order_date))
ORDER BY category_name, order_year, order_month;

-- 5. Lateral Joins / APPLY for top N products per category
SELECT
    c.category_name,
    top_products.product_id,
    top_products.product_name,
    top_products.total_sales
FROM categories c
CROSS APPLY (
    SELECT TOP 3
        p.product_id,
        p.product_name,
        SUM(o.total_amount) AS total_sales
    FROM products p
```

```
    INNER JOIN orders o ON o.product_id = p.product_id
    WHERE p.category_id = c.category_id
    GROUP BY p.product_id, p.product_name
    ORDER BY total_sales DESC
) AS top_products
ORDER BY c.category_name, top_products.total_sales DESC;

-- 6. Advanced windowing with cumulative metrics and moving averages
SELECT
    product_id,
    order_date,
    total_amount,
    SUM(total_amount) OVER (PARTITION BY product_id ORDER BY order_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) / 7.0 AS moving_avg_7_days,
    AVG(total_amount) OVER (PARTITION BY product_id ORDER BY order_date ROWS BETWEEN 29 PRECEDING AND CURRENT ROW) AS moving_avg_30_days
FROM orders
WHERE order_date BETWEEN '2024-01-01' AND '2024-12-31'
ORDER BY product_id, order_date;

-- 7. Performance-focused Index Hint and Query Optimization
SELECT /*+ INDEX(o idx_orders_customer_date) */
    o.customer_id,
    COUNT(o.order_id) AS total_orders,
    SUM(o.total_amount) AS total_spent
FROM orders o
```

```
WHERE o.order_date >= '2025-01-01'
GROUP BY o.customer_id
HAVING SUM(o.total_amount) > 1000
ORDER BY total_spent DESC;

-- 8. Anti-Pattern detection with exception handling in SQL
BEGIN TRY
    -- Detect inconsistent order amounts
    SELECT order_id, customer_id, total_amount
    FROM orders
    WHERE total_amount <= 0;
END TRY
BEGIN CATCH
    PRINT 'Warning: Found orders with zero or negative total_amount';
END CATCH;

-- 9. Combining rare advanced SQL constructs: PIVOT + JSON + Window Functions
WITH revenue_data AS (
    SELECT
        c.category_name,
        EXTRACT(MONTH FROM o.order_date) AS month,
        SUM(o.total_amount) AS monthly_revenue
    FROM orders o
    INNER JOIN products p ON o.product_id = p.product_id
    INNER JOIN categories c ON p.category_id = c.category_id
```

```
    GROUP BY c.category_name, EXTRACT(MONTH FROM o.order_date)
)
SELECT *
FROM revenue_data
PIVOT (
    SUM(monthly_revenue) FOR month IN ([1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12])
) AS pivoted_revenue
ORDER BY category_name;


-- 10. Full MAANG-style challenge: Combining all techniques
WITH recursive_customers AS (
    SELECT
        customer_id,
        1 AS depth
    FROM customers
    WHERE referral_id IS NULL
    UNION ALL
    SELECT
        c.customer_id,
        rc.depth + 1
    FROM customers c
    INNER JOIN recursive_customers rc
        ON c.referral_id = rc.customer_id
),
```

```
customer_metrics AS (
    SELECT
        c.customer_id,
        COUNT(o.order_id) AS total_orders,
        SUM(o.total_amount) AS total_spent,
        AVG(o.total_amount) AS avg_order_value,
        MAX(o.order_date) - MIN(o.order_date) AS active_days
    FROM recursive_customers rc
    LEFT JOIN orders o ON rc.customer_id = o.customer_id
    GROUP BY c.customer_id
)
SELECT
    cm.customer_id,
    cm.total_orders,
    cm.total_spent,
    cm.avg_order_value,
    cm.active_days,
    RANK() OVER (ORDER BY cm.total_spent DESC) AS lifetime_rank,
    DENSE_RANK() OVER (PARTITION BY FLOOR(cm.total_spent/500) ORDER BY cm.avg_order_value DESC) AS tier_rank
FROM customer_metrics cm
WHERE cm.total_orders > 5
ORDER BY lifetime_rank;
```