-- E-COMMERCE SALES ANALYTICS - COMPREHENSIVE SQL QUERIES

-- Database: PostgreSQL/MySQL Compatible

-- Description: Complete collection of SQL queries for e-commerce data analysis

-- 1. DATA EXPLORATION AND BASIC STATISTICS

-- 1.1 Dataset Overview

```
SELECT
  COUNT(*) as total_records,
  COUNT(DISTINCT customer_id) as unique_customers,
  COUNT(DISTINCT product_id) as unique_products,
  MIN(order_date) as earliest_order,
  MAX(order_date) as latest_order,
  SUM(total_amount) as total_revenue
FROM orders;
```

-- 1.2 Data Quality Check

```
SELECT
  'orders' as table_name,
  SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) as null_order_id,
  SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) as null_customer_id,
  SUM(CASE WHEN order_date IS NULL THEN 1 ELSE 0 END) as null_order_date,
  SUM(CASE WHEN total_amount IS NULL THEN 1 ELSE 0 END) as null_total_amount,
  SUM(CASE WHEN total_amount <= 0 THEN 1 ELSE 0 END) as invalid_amounts
FROM orders;
```

*-- **1.3 Revenue Distribution***

```
SELECT
  CASE
    WHEN total_amount < 50 THEN 'Low (0-50)'
```

```
    WHEN total_amount < 100 THEN 'Medium (50-100)'
    WHEN total_amount < 200 THEN 'High (100-200)'
    ELSE 'Premium (200+)'
  END as revenue_category,
  COUNT(*) as order_count,
  ROUND(AVG(total_amount), 2) as avg_order_value,
  SUM(total_amount) as total_revenue
FROM orders
GROUP BY
  CASE
    WHEN total_amount < 50 THEN 'Low (0-50)'
    WHEN total_amount < 100 THEN 'Medium (50-100)'
    WHEN total_amount < 200 THEN 'High (100-200)'
    ELSE 'Premium (200+)'
  END
ORDER BY avg_order_value;

-- 2. SALES TREND ANALYSIS

-- 2.1 Monthly Sales Trend
SELECT
  EXTRACT(YEAR FROM order_date) as year,
  EXTRACT(MONTH FROM order_date) as month,
  COUNT(*) as total_orders,
  SUM(total_amount) as monthly_revenue,
  ROUND(AVG(total_amount), 2) as avg_order_value,
  COUNT(DISTINCT customer_id) as unique_customers
FROM orders
GROUP BY EXTRACT(YEAR FROM order_date), EXTRACT(MONTH FROM order_date)
```

```
ORDER BY year, month;


-- 2.2 Day of Week Analysis
SELECT
  EXTRACT(DOW FROM order_date) as day_of_week,
  CASE EXTRACT(DOW FROM order_date)
    WHEN 0 THEN 'Sunday'
    WHEN 1 THEN 'Monday'
    WHEN 2 THEN 'Tuesday'
    WHEN 3 THEN 'Wednesday'
    WHEN 4 THEN 'Thursday'
    WHEN 5 THEN 'Friday'
    WHEN 6 THEN 'Saturday'
  END as day_name,
  COUNT(*) as total_orders,
  SUM(total_amount) as daily_revenue,
  ROUND(AVG(total_amount), 2) as avg_order_value
FROM orders
GROUP BY EXTRACT(DOW FROM order_date)
ORDER BY day_of_week;


-- 2.3 Quarterly Growth Analysis
WITH quarterly_sales AS (
  SELECT
    EXTRACT(YEAR FROM order_date) as year,
    EXTRACT(QUARTER FROM order_date) as quarter,
    SUM(total_amount) as revenue
  FROM orders
  GROUP BY EXTRACT(YEAR FROM order_date), EXTRACT(QUARTER FROM order_date)
```

```
),
quarterly_growth AS (
  SELECT
    year,
    quarter,
    revenue,
    LAG(revenue) OVER (ORDER BY year, quarter) as prev_quarter_revenue
  FROM quarterly_sales
)
SELECT
  year,
  quarter,
  revenue,
  prev_quarter_revenue,
  CASE
    WHEN prev_quarter_revenue IS NOT NULL
    THEN ROUND(((revenue - prev_quarter_revenue) / prev_quarter_revenue * 100), 2)
    ELSE NULL
  END as growth_percentage
FROM quarterly_growth
ORDER BY year, quarter;

-- 3. CUSTOMER ANALYSIS

-- 3.1 Top 20 Customers by Revenue
SELECT
  customer_id,
  COUNT(*) as total_orders,
  SUM(total_amount) as total_spent,
```

```
  ROUND(AVG(total_amount), 2) as avg_order_value,
  MIN(order_date) as first_order_date,
  MAX(order_date) as last_order_date,
  EXTRACT(DAYS FROM (MAX(order_date) - MIN(order_date))) as customer_lifespan_days
FROM orders
GROUP BY customer_id
ORDER BY total_spent DESC
LIMIT 20;

-- 3.2 Customer Segmentation (RFM Analysis)
WITH customer_rfm AS (
  SELECT
    customer_id,
    MAX(order_date) as last_order_date,
    COUNT(*) as frequency,
    SUM(total_amount) as monetary_value,
    EXTRACT(DAYS FROM (CURRENT_DATE - MAX(order_date))) as recency_days
  FROM orders
  GROUP BY customer_id
),
rfm_scored AS (
  SELECT
    customer_id,
    recency_days,
    frequency,
    monetary_value,
    NTILE(5) OVER (ORDER BY recency_days DESC) as recency_score,
    NTILE(5) OVER (ORDER BY frequency ASC) as frequency_score,
    NTILE(5) OVER (ORDER BY monetary_value ASC) as monetary_score
```

```
    FROM customer_rfm
)
SELECT
    CASE
        WHEN recency_score >= 4 AND frequency_score >= 4 AND monetary_score >= 4 THEN 'Champions'
        WHEN recency_score >= 3 AND frequency_score >= 3 AND monetary_score >= 3 THEN 'Loyal Customers'
        WHEN recency_score >= 3 AND frequency_score <= 2 AND monetary_score >= 3 THEN 'Potential
Loyalists'
        WHEN recency_score >= 4 AND frequency_score <= 2 AND monetary_score <= 2 THEN 'New Customers'
        WHEN recency_score <= 2 AND frequency_score >= 3 AND monetary_score >= 3 THEN 'At Risk'
        WHEN recency_score <= 2 AND frequency_score <= 2 AND monetary_score >= 3 THEN 'Cannot Lose
Them'
        WHEN recency_score <= 2 AND frequency_score <= 2 AND monetary_score <= 2 THEN 'Hibernating'
        ELSE 'Others'
    END as customer_segment,
    COUNT(*) as customer_count,
    ROUND(AVG(monetary_value), 2) as avg_monetary_value,
    ROUND(AVG(frequency), 2) as avg_frequency,
    ROUND(AVG(recency_days), 2) as avg_recency_days
FROM rfm_scored
GROUP BY
    CASE
        WHEN recency_score >= 4 AND frequency_score >= 4 AND monetary_score >= 4 THEN 'Champions'
        WHEN recency_score >= 3 AND frequency_score >= 3 AND monetary_score >= 3 THEN 'Loyal Customers'
        WHEN recency_score >= 3 AND frequency_score <= 2 AND monetary_score >= 3 THEN 'Potential
Loyalists'
        WHEN recency_score >= 4 AND frequency_score <= 2 AND monetary_score <= 2 THEN 'New Customers'
        WHEN recency_score <= 2 AND frequency_score >= 3 AND monetary_score >= 3 THEN 'At Risk'
        WHEN recency_score <= 2 AND frequency_score <= 2 AND monetary_score >= 3 THEN 'Cannot Lose
Them'
        WHEN recency_score <= 2 AND frequency_score <= 2 AND monetary_score <= 2 THEN 'Hibernating'
```

```
    ELSE 'Others'
  END
ORDER BY avg_monetary_value DESC;

-- 3.3 Customer Lifetime Value (CLV)
WITH customer_metrics AS (
  SELECT
    customer_id,
    COUNT(*) as total_orders,
    SUM(total_amount) as total_revenue,
    AVG(total_amount) as avg_order_value,
    MIN(order_date) as first_order,
    MAX(order_date) as last_order,
    EXTRACT(DAYS FROM (MAX(order_date) - MIN(order_date))) + 1 as customer_lifespan_days
  FROM orders
  GROUP BY customer_id
  HAVING COUNT(*) > 1  -- Only customers with multiple orders
)
SELECT
  customer_id,
  total_orders,
  total_revenue,
  avg_order_value,
  customer_lifespan_days,
  CASE
    WHEN customer_lifespan_days > 0
    THEN ROUND((total_orders::DECIMAL / customer_lifespan_days) * 365, 2)
    ELSE 0
  END as annual_order_frequency,
```

```
  CASE
    WHEN customer_lifespan_days > 0
    THEN ROUND(avg_order_value * (total_orders::DECIMAL / customer_lifespan_days) * 365, 2)
    ELSE total_revenue
  END as estimated_annual_clv
FROM customer_metrics
ORDER BY estimated_annual_clv DESC
LIMIT 50;

-- 4. PRODUCT PERFORMANCE ANALYSIS

-- 4.1 Top Selling Products
SELECT
  product_id,
  product_name,
  product_category,
  SUM(quantity) as total_quantity_sold,
  COUNT(*) as times_ordered,
  SUM(total_amount) as total_revenue,
  ROUND(AVG(unit_price), 2) as avg_unit_price,
  ROUND(SUM(total_amount) / SUM(quantity), 2) as revenue_per_unit
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY product_id, product_name, product_category
ORDER BY total_revenue DESC
LIMIT 20;

-- 4.2 Category Performance Analysis
```

```
SELECT
  product_category,
  COUNT(DISTINCT product_id) as unique_products,
  SUM(quantity) as total_quantity_sold,
  SUM(total_amount) as category_revenue,
  ROUND(AVG(unit_price), 2) as avg_product_price,
  COUNT(*) as total_orders,
  COUNT(DISTINCT customer_id) as unique_customers,
  ROUND(SUM(total_amount) / COUNT(DISTINCT customer_id), 2) as revenue_per_customer
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY product_category
ORDER BY category_revenue DESC;

-- 4.3 Product Performance by Season
SELECT
  product_category,
  CASE
    WHEN EXTRACT(MONTH FROM order_date) IN (12, 1, 2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM order_date) IN (3, 4, 5) THEN 'Spring'
    WHEN EXTRACT(MONTH FROM order_date) IN (6, 7, 8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM order_date) IN (9, 10, 11) THEN 'Fall'
  END as season,
  SUM(quantity) as total_quantity,
  SUM(total_amount) as seasonal_revenue,
  COUNT(DISTINCT customer_id) as unique_customers
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
```

```
JOIN products p ON oi.product_id = p.product_id
GROUP BY product_category,
  CASE
    WHEN EXTRACT(MONTH FROM order_date) IN (12, 1, 2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM order_date) IN (3, 4, 5) THEN 'Spring'
    WHEN EXTRACT(MONTH FROM order_date) IN (6, 7, 8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM order_date) IN (9, 10, 11) THEN 'Fall'
  END
ORDER BY product_category, seasonal_revenue DESC;

-- 5. ADVANCED ANALYTICS

-- 5.1 Cohort Analysis for Customer Retention
WITH customer_cohorts AS (
  SELECT
    customer_id,
    MIN(DATE_TRUNC('month', order_date)) as cohort_month
  FROM orders
  GROUP BY customer_id
),
customer_activities AS (
  SELECT
    o.customer_id,
    cc.cohort_month,
    DATE_TRUNC('month', o.order_date) as order_month,
    EXTRACT(MONTH FROM AGE(o.order_date, cc.cohort_month)) as month_number
  FROM orders o
  JOIN customer_cohorts cc ON o.customer_id = cc.customer_id
),
```

```
cohort_table AS (
  SELECT
    cohort_month,
    month_number,
    COUNT(DISTINCT customer_id) as customers
  FROM customer_activities
  GROUP BY cohort_month, month_number
),
cohort_sizes AS (
  SELECT
    cohort_month,
    COUNT(DISTINCT customer_id) as cohort_size
  FROM customer_cohorts
  GROUP BY cohort_month
)
SELECT
  ct.cohort_month,
  cs.cohort_size,
  ct.month_number,
  ct.customers,
  ROUND((ct.customers::DECIMAL / cs.cohort_size) * 100, 2) as retention_rate
FROM cohort_table ct
JOIN cohort_sizes cs ON ct.cohort_month = cs.cohort_month
ORDER BY ct.cohort_month, ct.month_number;

-- 5.2 Market Basket Analysis
WITH order_products AS (
  SELECT DISTINCT
    o.order_id,
```

```
    p.product_name,
    p.product_category
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  JOIN products p ON oi.product_id = p.product_id
),
product_pairs AS (
  SELECT
    op1.product_name as product_a,
    op2.product_name as product_b,
    COUNT(*) as frequency
  FROM order_products op1
  JOIN order_products op2 ON op1.order_id = op2.order_id
  WHERE op1.product_name < op2.product_name  -- Avoid duplicates
  GROUP BY op1.product_name, op2.product_name
  HAVING COUNT(*) >= 5  -- Only pairs that appear together at least 5 times
),
product_frequencies AS (
  SELECT
    product_name,
    COUNT(DISTINCT order_id) as individual_frequency
  FROM order_products
  GROUP BY product_name
)
SELECT
  pp.product_a,
  pp.product_b,
  pp.frequency as together_frequency,
  pf1.individual_frequency as product_a_frequency,
```

```
    pf2.individual_frequency as product_b_frequency,
    ROUND((pp.frequency::DECIMAL / pf1.individual_frequency) * 100, 2) as confidence_a_to_b,
    ROUND((pp.frequency::DECIMAL / pf2.individual_frequency) * 100, 2) as confidence_b_to_a
FROM product_pairs pp
JOIN product_frequencies pf1 ON pp.product_a = pf1.product_name
JOIN product_frequencies pf2 ON pp.product_b = pf2.product_name
ORDER BY pp.frequency DESC, confidence_a_to_b DESC
LIMIT 20;

-- 6. GEOGRAPHIC ANALYSIS

-- 6.1 Sales by Geographic Location
SELECT
    customer_city,
    customer_state,
    COUNT(DISTINCT customer_id) as unique_customers,
    COUNT(*) as total_orders,
    SUM(total_amount) as total_revenue,
    ROUND(AVG(total_amount), 2) as avg_order_value,
    ROUND(SUM(total_amount) / COUNT(DISTINCT customer_id), 2) as revenue_per_customer
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY customer_city, customer_state
HAVING COUNT(*) >= 10  -- Only cities with at least 10 orders
ORDER BY total_revenue DESC
LIMIT 20;

-- 6.2 Regional Performance Comparison
SELECT
```

```
  CASE
    WHEN customer_state IN ('CA', 'OR', 'WA') THEN 'West'
    WHEN customer_state IN ('TX', 'AZ', 'NM', 'OK') THEN 'Southwest'
    WHEN customer_state IN ('NY', 'NJ', 'CT', 'MA', 'PA') THEN 'Northeast'
    WHEN customer_state IN ('FL', 'GA', 'SC', 'NC', 'VA') THEN 'Southeast'
    ELSE 'Other'
  END as region,
  COUNT(DISTINCT o.customer_id) as unique_customers,
  COUNT(*) as total_orders,
  SUM(total_amount) as total_revenue,
  ROUND(AVG(total_amount), 2) as avg_order_value,
  ROUND(SUM(total_amount) / COUNT(*), 2) as revenue_per_order
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY
  CASE
    WHEN customer_state IN ('CA', 'OR', 'WA') THEN 'West'
    WHEN customer_state IN ('TX', 'AZ', 'NM', 'OK') THEN 'Southwest'
    WHEN customer_state IN ('NY', 'NJ', 'CT', 'MA', 'PA') THEN 'Northeast'
    WHEN customer_state IN ('FL', 'GA', 'SC', 'NC', 'VA') THEN 'Southeast'
    ELSE 'Other'
  END
ORDER BY total_revenue DESC;
```

-- 7. PAYMENT AND SHIPPING ANALYSIS

-- 7.1 Payment Method Performance

```
SELECT
  payment_method,
  COUNT(*) as transaction_count,
```

```
    SUM(total_amount) as total_revenue,
    ROUND(AVG(total_amount), 2) as avg_transaction_value,
    COUNT(DISTINCT customer_id) as unique_customers,
    ROUND((COUNT(*) * 100.0) / SUM(COUNT(*)) OVER (), 2) as percentage_of_orders
FROM orders
GROUP BY payment_method
ORDER BY total_revenue DESC;

-- 7.2 Shipping Analysis
SELECT
    shipping_method,
    COUNT(*) as order_count,
    SUM(total_amount) as total_revenue,
    ROUND(AVG(total_amount), 2) as avg_order_value,
    ROUND(AVG(shipping_cost), 2) as avg_shipping_cost,
    ROUND(AVG(EXTRACT(DAYS FROM (shipped_date - order_date))), 1) as avg_processing_days
FROM orders
WHERE shipped_date IS NOT NULL
GROUP BY shipping_method
ORDER BY order_count DESC;

-- 8. CUSTOMER DEMOGRAPHICS ANALYSIS

-- 8.1 Age Group Analysis
SELECT
    CASE
        WHEN customer_age < 25 THEN '18-24'
        WHEN customer_age < 35 THEN '25-34'
        WHEN customer_age < 45 THEN '35-44'
```

```
    WHEN customer_age < 55 THEN '45-54'
    WHEN customer_age < 65 THEN '55-64'
    ELSE '65+'
  END as age_group,
  COUNT(DISTINCT o.customer_id) as unique_customers,
  COUNT(*) as total_orders,
  SUM(total_amount) as total_revenue,
  ROUND(AVG(total_amount), 2) as avg_order_value,
  ROUND(COUNT(*) / COUNT(DISTINCT o.customer_id), 2) as avg_orders_per_customer
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE customer_age IS NOT NULL
GROUP BY
  CASE
    WHEN customer_age < 25 THEN '18-24'
    WHEN customer_age < 35 THEN '25-34'
    WHEN customer_age < 45 THEN '35-44'
    WHEN customer_age < 55 THEN '45-54'
    WHEN customer_age < 65 THEN '55-64'
    ELSE '65+'
  END
ORDER BY avg_order_value DESC;

-- 8.2 Gender-based Purchasing Patterns
SELECT
  c.customer_gender,
  COUNT(DISTINCT o.customer_id) as unique_customers,
  COUNT(*) as total_orders,
  SUM(o.total_amount) as total_revenue,
```

```
    ROUND(AVG(o.total_amount), 2) as avg_order_value,
    STRING_AGG(DISTINCT p.product_category, ', ' ORDER BY p.product_category) as popular_categories
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
WHERE c.customer_gender IN ('Male', 'Female')
GROUP BY c.customer_gender
ORDER BY total_revenue DESC;

-- 9. PERFORMANCE METRICS AND KPIs

-- 9.1 Key Business Metrics Summary
WITH metrics AS (
    SELECT
        COUNT(DISTINCT customer_id) as total_customers,
        COUNT(*) as total_orders,
        SUM(total_amount) as total_revenue,
        AVG(total_amount) as avg_order_value,
        COUNT(*) / COUNT(DISTINCT customer_id) as avg_orders_per_customer,
        MIN(order_date) as first_order_date,
        MAX(order_date) as last_order_date
    FROM orders
)
SELECT
    total_customers,
    total_orders,
    ROUND(total_revenue, 2) as total_revenue,
    ROUND(avg_order_value, 2) as avg_order_value,
```

```
    ROUND(avg_orders_per_customer, 2) as avg_orders_per_customer,
    first_order_date,
    last_order_date,
    EXTRACT(DAYS FROM (last_order_date - first_order_date)) as business_days,
    ROUND(total_revenue / EXTRACT(DAYS FROM (last_order_date - first_order_date)), 2) as daily_avg_revenue
FROM metrics;

-- END OF SQL QUERIES COLLECTION
```