Advanced SQL Sample for E-Commerce Analytics

Scenario

A tech e-commerce company wants to:

1. Track user sessions across web and mobile.
2. Analyze product performance.
3. Detect anomalies in transaction behavior.
4. Implement RBAC for analytics teams.
5. Optimize queries with materialized views and partitioning.

Extreme, Advanced, SQL

```
-- 1. Create base tables

CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE products (
    product_id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    category TEXT,
    price NUMERIC(10, 2),
```

```
    stock_quantity INTEGER,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE sessions (
    session_id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(user_id),
    session_start TIMESTAMP,
    session_end TIMESTAMP,
    device_type TEXT CHECK (device_type IN ('web', 'mobile')),
    ip_address INET
);

CREATE TABLE page_views (
    view_id UUID PRIMARY KEY,
    session_id UUID REFERENCES sessions(session_id),
    product_id UUID REFERENCES products(product_id),
    viewed_at TIMESTAMP
);

CREATE TABLE transactions (
    transaction_id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(user_id),
    product_id UUID REFERENCES products(product_id),
```

```
    session_id UUID REFERENCES sessions(session_id),
    quantity INTEGER CHECK (quantity > 0),
    total_amount NUMERIC(10, 2),
    status TEXT CHECK (status IN ('success', 'failed', 'refunded')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Add indexes to optimize performance

CREATE INDEX idx_transactions_user_time ON transactions (user_id, created_at);
CREATE INDEX idx_sessions_user_time ON sessions (user_id, session_start);
CREATE INDEX idx_page_views_product_time ON page_views (product_id,
viewed_at);
CREATE INDEX idx_products_category_price ON products (category, price);

-- 3. Insert dummy data for analytics (normally done via ETL or scripts)

-- 4. Build views for analytics

-- 4.1 Session Duration View
CREATE VIEW user_session_durations AS
SELECT
    user_id,
    session_id,
    device_type,
```

```
    session_end - session_start AS duration,
    DATE(session_start) AS session_date
FROM sessions
WHERE session_end IS NOT NULL;

-- 4.2 Product Conversion Funnel View
CREATE VIEW product_conversion_funnel AS
SELECT
    p.product_id,
    p.name,
    COUNT(DISTINCT pv.view_id) AS views,
    COUNT(DISTINCT t.transaction_id) FILTER (WHERE t.status = 'success') AS
purchases,
    ROUND(
        (COUNT(DISTINCT t.transaction_id) FILTER (WHERE t.status =
'success')::NUMERIC /
         NULLIF(COUNT(DISTINCT pv.view_id), 0)) * 100, 2
    ) AS conversion_rate
FROM products p
LEFT JOIN page_views pv ON pv.product_id = p.product_id
LEFT JOIN transactions t ON t.product_id = p.product_id
GROUP BY p.product_id, p.name;

-- 4.3 Daily Revenue View (Materialized)
CREATE MATERIALIZED VIEW daily_revenue_summary AS
SELECT
```

```
    DATE(created_at) AS revenue_date,
    SUM(total_amount) FILTER (WHERE status = 'success') AS total_revenue,
    COUNT(DISTINCT transaction_id) FILTER (WHERE status = 'success') AS
successful_orders,
    COUNT(DISTINCT transaction_id) FILTER (WHERE status = 'refunded') AS refunds
FROM transactions
GROUP BY DATE(created_at);


-- 5. Anomaly Detection: Find users with sudden spikes in spending


WITH user_daily_spend AS (
    SELECT
        user_id,
        DATE(created_at) AS day,
        SUM(total_amount) AS total_spent
    FROM transactions
    WHERE status = 'success'
    GROUP BY user_id, DATE(created_at)
),
user_spend_stats AS (
    SELECT
        user_id,
        AVG(total_spent) AS avg_spent,
        STDDEV_POP(total_spent) AS std_dev_spent
    FROM user_daily_spend
```

```
    GROUP BY user_id
)
SELECT
    d.user_id,
    d.day,
    d.total_spent,
    s.avg_spent,
    s.std_dev_spent,
    (d.total_spent - s.avg_spent) / NULLIF(s.std_dev_spent, 0) AS z_score
FROM user_daily_spend d
JOIN user_spend_stats s ON d.user_id = s.user_id
WHERE (d.total_spent - s.avg_spent) / NULLIF(s.std_dev_spent, 0) > 3.0 -- 3σ rule
ORDER BY z_score DESC;

-- 6. Recursive Query: User Referral Tree

CREATE TABLE user_referrals (
    user_id UUID PRIMARY KEY,
    referred_by UUID REFERENCES users(user_id)
);

-- Recursive CTE to build referral hierarchy
WITH RECURSIVE referral_tree AS (
    SELECT
        u.user_id,
```

```
    u.name,
    NULL::UUID AS root_referrer,
    0 AS level
  FROM users u
  WHERE u.user_id NOT IN (SELECT referred_by FROM user_referrals WHERE referred_by IS NOT NULL)

  UNION ALL

  SELECT
    ur.user_id,
    u.name,
    rt.user_id AS root_referrer,
    rt.level + 1
  FROM user_referrals ur
  JOIN users u ON u.user_id = ur.user_id
  JOIN referral_tree rt ON rt.user_id = ur.referred_by
)
SELECT * FROM referral_tree;

-- 7. Partitioned Table for Transactions (if DB supports it)

CREATE TABLE transactions_partitioned (
  transaction_id UUID NOT NULL,
  user_id UUID NOT NULL,
```

```
    product_id UUID NOT NULL,
    session_id UUID,
    quantity INTEGER NOT NULL,
    total_amount NUMERIC(10,2),
    status TEXT,
    created_at TIMESTAMP NOT NULL
) PARTITION BY RANGE (created_at);

CREATE TABLE transactions_2024_q1 PARTITION OF transactions_partitioned
    FOR VALUES FROM ('2024-01-01') TO ('2024-04-01');

CREATE TABLE transactions_2024_q2 PARTITION OF transactions_partitioned
    FOR VALUES FROM ('2024-04-01') TO ('2024-07-01');

-- 8. RBAC (Role-Based Access Control)

-- Create roles
CREATE ROLE analyst;
CREATE ROLE admin;

-- Grant access
GRANT SELECT ON ALL TABLES IN SCHEMA public TO analyst;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA
public TO admin;
```

```
-- Add user and assign roles
CREATE USER analyst_user WITH PASSWORD 'securepassword';
GRANT analyst TO analyst_user;

-- 9. Refresh Materialized Views (to be done via job scheduler)
REFRESH MATERIALIZED VIEW daily_revenue_summary;

-- 10. Audit Logging (Optional, for compliance)
CREATE TABLE audit_log (
    log_id SERIAL PRIMARY KEY,
    event_type TEXT,
    user_id UUID,
    description TEXT,
    event_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE FUNCTION log_transaction_change() RETURNS TRIGGER
AS $$
BEGIN
    INSERT INTO audit_log(event_type, user_id, description)
    VALUES (
        TG_OP,
        NEW.user_id,
        format('Transaction %s: %s', TG_OP, NEW.transaction_id)
    );
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_transaction_log
AFTER INSERT OR UPDATE OR DELETE ON transactions
FOR EACH ROW EXECUTE FUNCTION log_transaction_change();

-- Done!
```