Enterprise SQL Sample: Scalable E-Commerce Analytics Platform

Scenario

You’re building a backend analytics system for a global e-commerce platform. The system must support complex reporting, multi-language product search, JSON-based cart tracking, and role-based access.

1. Schema Design

```
-- Main Users Table
CREATE TABLE users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email TEXT UNIQUE NOT NULL,
    full_name TEXT,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    is_active BOOLEAN DEFAULT TRUE
);

-- Products Table with Multi-language Support
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    sku TEXT UNIQUE NOT NULL,
    price NUMERIC(10, 2) NOT NULL,
    available_stock INT DEFAULT 0,
    attributes JSONB DEFAULT '{}',
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Product Translations for Localization
CREATE TABLE product_translations (
    product_id INT REFERENCES products(product_id) ON DELETE CASCADE,
    locale TEXT NOT NULL,
    title TEXT NOT NULL,
    description TEXT,
    PRIMARY KEY (product_id, locale)
);

-- Orders and Line Items
CREATE TABLE orders (
    order_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(user_id),
    order_status TEXT CHECK (order_status IN ('pending', 'shipped', 'delivered',
'cancelled')),
```

```
    total_amount NUMERIC(10, 2) NOT NULL,
    ordered_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE order_items (
    order_item_id SERIAL PRIMARY KEY,
    order_id UUID REFERENCES orders(order_id) ON DELETE CASCADE,
    product_id INT REFERENCES products(product_id),
    quantity INT CHECK (quantity > 0),
    unit_price NUMERIC(10, 2) NOT NULL
);

-- Shopping Cart (JSON format for flexibility)
CREATE TABLE shopping_carts (
    user_id UUID PRIMARY KEY REFERENCES users(user_id) ON DELETE
CASCADE,
    cart_data JSONB NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

2. Indexes for Performance

```
CREATE INDEX idx_shopping_cart_data ON shopping_carts USING GIN (cart_data);
CREATE INDEX idx_product_translations_fts ON product_translations
    USING GIN (to_tsvector('simple', title || ' ' || description));
CREATE INDEX idx_orders_user_id ON orders(user_id);
CREATE INDEX idx_order_items_product_id ON order_items(product_id);
```

3. Recursive CTE: Order Hierarchies

```
CREATE TABLE order_referrals (
    parent_order UUID REFERENCES orders(order_id),
    child_order UUID REFERENCES orders(order_id),
    PRIMARY KEY (parent_order, child_order)
);

WITH RECURSIVE order_tree AS (
    SELECT parent_order, child_order
    FROM order_referrals
    WHERE parent_order = 'root-order-uuid'
    UNION ALL
    SELECT r.parent_order, o.child_order
    FROM order_tree o
    JOIN order_referrals r ON o.child_order = r.parent_order
)
SELECT * FROM order_tree;
```

4. Window Functions: Lifetime Value (LTV) per User

```
SELECT
   user_id,
   SUM(total_amount) AS lifetime_value,
   RANK() OVER (ORDER BY SUM(total_amount) DESC) AS user_rank
FROM orders
GROUP BY user_id;
```

5. JSON Querying: Cart Insights

```
SELECT
   product_id,
   COUNT(*) AS frequency
FROM (
   SELECT jsonb_object_keys(cart_data) AS product_id
   FROM shopping_carts
) sub
GROUP BY product_id
ORDER BY frequency DESC
LIMIT 5;
```

6. Full-Text Search: Product Search

```
SELECT
   p.product_id,
   pt.locale,
   pt.title,
   ts_rank(to_tsvector(pt.title || ' ' || pt.description), plainto_tsquery('english', 'wireless
charger')) AS rank
FROM product_translations pt
JOIN products p ON p.product_id = pt.product_id
WHERE to_tsvector(pt.title || ' ' || pt.description) @@ plainto_tsquery('english', 'wireless
charger')
ORDER BY rank DESC
LIMIT 10;
```

7. Materialized View for Reporting

```
CREATE MATERIALIZED VIEW mv_monthly_sales AS
SELECT
   DATE_TRUNC('month', ordered_at) AS order_month,
   product_id,
   SUM(quantity) AS total_quantity,
   SUM(quantity * unit_price) AS total_revenue
FROM orders o
```

```
JOIN order_items oi ON o.order_id = oi.order_id
WHERE order_status = 'delivered'
GROUP BY order_month, product_id;
```

8. Role-Based Access Control (RBAC)

```
CREATE ROLE admin NOINHERIT;
CREATE ROLE analyst;
CREATE ROLE customer;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA
public TO admin;
GRANT SELECT ON mv_monthly_sales TO analyst;
GRANT SELECT, UPDATE ON shopping_carts TO customer;

GRANT customer TO some_user;
```

9. Partitioning for Scalability

```
CREATE TABLE orders_partitioned (
   order_id UUID,
   user_id UUID,
   order_status TEXT,
   total_amount NUMERIC(10, 2),
   ordered_at TIMESTAMPTZ
) PARTITION BY RANGE (ordered_at);

CREATE TABLE orders_2025_01 PARTITION OF orders_partitioned
   FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE orders_2025_02 PARTITION OF orders_partitioned
   FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');
```

10. Security & Auditing Example

```
CREATE TABLE price_audit (
   product_id INT,
   old_price NUMERIC(10,2),
   new_price NUMERIC(10,2),
   changed_by TEXT,
   changed_at TIMESTAMPTZ DEFAULT now()
);

CREATE OR REPLACE FUNCTION log_price_change() RETURNS TRIGGER AS $$
BEGIN
   IF OLD.price IS DISTINCT FROM NEW.price THEN
      INSERT INTO price_audit(product_id, old_price, new_price, changed_by)
      VALUES (OLD.product_id, OLD.price, NEW.price, current_user);
   END IF;
```

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

CREATE TRIGGER trg_log_price_change
BEFORE UPDATE ON products
FOR EACH ROW
EXECUTE FUNCTION log_price_change();
```