Stored Procedures - In-Class Task #6

Database Setup

First, create the following tables and insert sample data:

sql
```sql
-- Create categories table
CREATE TABLE categories (
    category_id SERIAL PRIMARY KEY,
    category_name VARCHAR(50),
    description TEXT
);

-- Insert sample data
INSERT INTO categories (category_name, description) VALUES
('Electronics', 'Computers, phones, and gadgets'),
('Clothing', 'Fashion and apparel'),
('Books', 'Physical and digital books'),
('Home & Kitchen', 'Household items and appliances'),
('Sports', 'Sports equipment and accessories');

-- Create products table
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100),
    category_id INTEGER REFERENCES categories(category_id),
    price NUMERIC(10,2),
    stock_quantity INTEGER,
    rating NUMERIC(2,1),
    vendor VARCHAR(100)
);

-- Insert sample data
INSERT INTO products (product_name, category_id, price, stock_quantity, rating, vendor)
VALUES
('iPhone 15 Pro', 1, 599990, 25, 4.8, 'Apple Store KZ'),
('Samsung Galaxy S24', 1, 499990, 30, 4.7, 'Samsung Kazakhstan'),
('MacBook Air M2', 1, 899990, 15, 4.9, 'Apple Store KZ'),
('Wireless Mouse', 1, 12990, 100, 4.5, 'Logitech Kazakhstan'),
('Mechanical Keyboard', 1, 45990, 50, 4.6, 'Keychron Asia'),
('Winter Jacket', 2, 89990, 40, 4.4, 'Zara Kazakhstan'),
('Running Shoes', 2, 54990, 60, 4.7, 'Nike Almaty'),
('Cotton T-Shirt', 2, 8990, 150, 4.3, 'H&M Kazakhstan'),
('Database Systems Book', 3, 15990, 80, 4.9, 'Meloman Books'),
('Clean Code', 3, 18990, 45, 4.8, 'Meloman Books'),
('Coffee Maker', 4, 69990, 20, 4.5, 'Philips KZ'),
('Blender', 4, 34990, 35, 4.4, 'Bosch Kazakhstan'),
('Yoga Mat', 5, 12990, 70, 4.6, 'Decathlon Almaty'),
('Dumbbells Set', 5, 45990, 25, 4.7, 'Sportmaster KZ');

-- Create customers table
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    full_name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(20),
    city VARCHAR(50),
    loyalty_points INTEGER,
    registration_date DATE
);

-- Insert sample data
INSERT INTO customers (full_name, email, phone, city, loyalty_points, registration_date)
VALUES
('Әлішер Нұрғожин', 'alisher.n@mail.kz', '+77011234567', 'Almaty', 1250, '2023-01-15'),
('Балжан Сериккызы', 'balzhan.s@gmail.com', '+77012345678', 'Astana', 890, '2023-03-20'),
('Дамир Төлеуов', 'damir.t@inbox.kz', '+77023456789', 'Almaty', 2100, '2022-11-05'),
('Жанель Қайратқызы', 'zhanel.k@mail.kz', '+77034567890', 'Shymkent', 450, '2023-06-10'),
('Ернар Бекжанов', 'ernar.b@gmail.com', '+77045678901', 'Almaty', 3500, '2022-08-15'),
('Сандуғаш Амантай', 'sandugash.a@inbox.kz', '+77056789012', 'Karaganda', 670, '2023-04-22'),
('Нұрсұлтан Әлімов', 'nursultan.a@mail.kz', '+77067890123', 'Almaty', 1800, '2023-02-18'),
('Айым Жұмабекова', 'aiym.zh@gmail.com', '+77078901234', 'Astana', 950, '2023-05-30');

-- Create orders table
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INTEGER REFERENCES customers(customer_id),
    order_date TIMESTAMP,
    total_amount NUMERIC(12,2),
    discount_amount NUMERIC(10,2),
    status VARCHAR(20),
    payment_method VARCHAR(20),
    delivery_address TEXT
);

-- Insert sample data
INSERT INTO orders (customer_id, order_date, total_amount, discount_amount, status,
payment_method, delivery_address) VALUES
(1, '2024-11-01 10:30:00', 612980.00, 12990.00, 'delivered', 'card', 'Almaty, Dostyk Ave, 123'),
(1, '2024-11-15 14:20:00', 45990.00, 0.00, 'processing', 'card', 'Almaty, Dostyk Ave, 123'),
(2, '2024-11-02 09:15:00', 899990.00, 0.00, 'delivered', 'kaspi', 'Astana, Mangilik El, 55'),
(3, '2024-11-03 16:45:00', 138970.00, 8990.00, 'delivered', 'card', 'Almaty, Furmanov St, 88'),
(4, '2024-11-05 11:00:00', 54990.00, 0.00, 'shipped', 'cash', 'Shymkent, Baidibek Bi, 12'),
(5, '2024-11-06 13:30:00', 154980.00, 15990.00, 'delivered', 'kaspi', 'Almaty, Abai Ave, 45'),
(6, '2024-11-08 10:00:00', 69990.00, 0.00, 'delivered', 'card', 'Karaganda, Bukhar Zhyrau, 23'),
(7, '2024-11-10 15:45:00', 34990.00, 0.00, 'delivered', 'kaspi', 'Almaty, Satpaev St, 90'),
(8, '2024-11-12 12:20:00', 108980.00, 12990.00, 'processing', 'card', 'Astana, Kabanbay Batyr, 14'),
(3, '2024-11-14 09:30:00', 45990.00, 0.00, 'shipped', 'card', 'Almaty, Furmanov St, 88');
```

```sql
-- Create order_items table
CREATE TABLE order_items (
    item_id SERIAL PRIMARY KEY,
    order_id INTEGER REFERENCES orders(order_id),
    product_id INTEGER REFERENCES products(product_id),
    quantity INTEGER,
    price_at_purchase NUMERIC(10,2)
);

-- Insert sample data
INSERT INTO order_items (order_id, product_id, quantity, price_at_purchase) VALUES
(1, 1, 1, 599990.00),
(1, 4, 1, 12990.00),
(2, 5, 1, 45990.00),
(3, 3, 1, 899990.00),
(4, 6, 1, 89990.00),
(4, 7, 1, 54990.00),
(5, 7, 1, 54990.00),
(6, 9, 1, 15990.00),
(6, 10, 1, 18990.00),
(6, 3, 1, 899990.00),
(7, 11, 1, 69990.00),
(8, 12, 1, 34990.00),
(9, 6, 1, 89990.00),
(9, 13, 1, 12990.00),
(10, 14, 1, 45990.00);

-- Create reviews table
CREATE TABLE reviews (
    review_id SERIAL PRIMARY KEY,
    product_id INTEGER REFERENCES products(product_id),
    customer_id INTEGER REFERENCES customers(customer_id),
    rating INTEGER,
    review_text TEXT,
    review_date DATE
);

-- Insert sample data
INSERT INTO reviews (product_id, customer_id, rating, review_text, review_date) VALUES
(1, 1, 5, 'Отличный телефон! Камера просто супер!', '2024-11-05'),
(3, 2, 5, 'Лучший ноутбук для работы и учебы', '2024-11-08'),
(7, 4, 5, 'Очень удобные кроссовки', '2024-11-12'),
(9, 5, 5, 'Must-have книга для всех программистов', '2024-11-15'),
(11, 6, 4, 'Хорошая кофеварка, но немного шумная', '2024-11-14'),
(6, 3, 4, 'Качественная куртка, но маломерит', '2024-11-10');
```

Task 1: Price Calculation Functions (4 minutes)

Exercise 1.1: Dynamic Discount Calculator

Create a function called `calculate_dynamic_discount` that:

- Accepts parameters:
  - `order_total` (NUMERIC)
  - `loyalty_points` (INTEGER)
- Returns discount amount as NUMERIC
- Logic:
  - Base discount: 0%
  - If order_total > 100000: 5% discount
  - If order_total > 500000: 10% discount
  - Additional: 1% for every 1000 loyalty points (max 10% additional)
- Return: order_total × (base_discount + loyalty_discount) / 100

Test your function:

sql
```
SELECT calculate_dynamic_discount(150000, 500);  -- 5% + 0.5% = 5.5%
SELECT calculate_dynamic_discount(600000, 2500); -- 10% + 2.5% = 12.5%
SELECT calculate_dynamic_discount(80000, 1000);  -- 0% + 1% = 1%
```

Exercise 1.2: Shipping Cost Calculator

Create a function called `calculate_shipping_cost` that:

- Accepts parameters:
  - `order_total` (NUMERIC)
  - `city` (VARCHAR) with DEFAULT 'Almaty'
- Returns shipping cost as NUMERIC
- Logic:
  - Free shipping if order_total >= 50000
  - Almaty/Astana: 2000 KZT
  - Other cities: 3500 KZT

Test your function:

sql
```
SELECT calculate_shipping_cost(30000, 'Almaty');     -- 2000
SELECT calculate_shipping_cost(60000, 'Shymkent');   -- 0 (free)
SELECT calculate_shipping_cost(25000, 'Karaganda'); -- 3500
```

Task 2: Analytics Functions with OUT Parameters (4 minutes)

Exercise 2.1: Customer Purchase Statistics

Create a function called `customer_purchase_stats` that:

- Accepts one IN parameter: `p_customer_id` (INTEGER)
- Returns multiple OUT parameters:
  - `total_orders` (INTEGER) - count of orders

    - `total_spent` (NUMERIC) - sum of (total_amount - discount_amount)
    - `avg_order_value` (NUMERIC) - average order amount
    - `loyalty_tier` (VARCHAR) - 'Bronze'/'Silver'/'Gold'/'Platinum' based on total_spent
        - < 500000: Bronze
        - 500000-1000000: Silver
        - 1000000-2000000: Gold
        - 2000000: Platinum

Test your function:

```sql
SELECT * FROM customer_purchase_stats(1);
SELECT * FROM customer_purchase_stats(5);
```

Exercise 2.2: Product Performance Analysis

Create a function called `product_performance` that:

- Accepts one IN parameter: `p_product_id` (INTEGER)
- Returns OUT parameters:
    - `total_sold` (INTEGER) - sum of quantities from order_items
    - `revenue` (NUMERIC) - sum of (quantity × price_at_purchase)
    - `avg_rating` (NUMERIC) - average from reviews
    - `review_count` (INTEGER) - count of reviews
    - `current_stock` (INTEGER) - current stock_quantity

Test your function:

```sql
SELECT * FROM product_performance(1);
SELECT * FROM product_performance(7);
```

Task 3: Data Transformation with INOUT (3 minutes)

Exercise 3.1: Apply Loyalty Points

Create a function called `apply_loyalty_discount` that:

- Accepts INOUT parameter: `order_amount` (NUMERIC)
- Accepts IN parameters:
    - `loyalty_points` (INTEGER)
    - `points_to_use` (INTEGER)
- Conversion: 100 points = 1000 KZT discount
- Reduce order_amount by discount (but not below 0)
- Store new amount in `order_amount` parameter

Test your function:

sql
```sql
SELECT apply_loyalty_discount(50000, 2000, 1000);  -- Use 1000 points = 10000 discount
SELECT apply_loyalty_discount(15000, 5000, 2000);  -- Use 2000 points = 20000 discount (min 0)
```

Exercise 3.2: Category Discount Multiplier

Create a function called `apply_category_multiplier` that:

- Accepts INOUT parameter: `base_discount` (NUMERIC)
- Accepts IN parameter: `p_category_id` (INTEGER)
- Multiplies discount based on category:
  - Electronics (1): 1.2× (20% bonus)
  - Clothing (2): 1.5× (50% bonus)
  - Books (3): 1.1× (10% bonus)
  - Home & Kitchen (4): 1.3× (30% bonus)
  - Sports (5): 1.4× (40% bonus)

Test your function:

sql
```sql
SELECT apply_category_multiplier(10.0, 1);  -- 10 × 1.2 = 12
SELECT apply_category_multiplier(10.0, 2);  -- 10 × 1.5 = 15
```

Task 4: Table-Returning Functions (4 minutes)

Exercise 4.1: Customer Order History

Create a function called `get_customer_orders` that:

- Accepts parameters:
  - `p_customer_id` (INTEGER)
  - `p_status` (VARCHAR) with DEFAULT NULL
- Returns a table with columns:
  - `order_id` (INTEGER)
  - `order_date` (TIMESTAMP)
  - `total_amount` (NUMERIC)
  - `discount_amount` (NUMERIC)
  - `status` (VARCHAR)
  - `payment_method` (VARCHAR)
- If p_status IS NULL, return all orders
- Otherwise, filter by status

Test your function:

sql

```sql
SELECT * FROM get_customer_orders(1);                    -- All orders
SELECT * FROM get_customer_orders(3, 'delivered');       -- Only delivered
```