

Step 3 – PL/SQL Code

Create OrderDetailsType which will be used in the first function.

```
CREATE OR REPLACE TYPE OrderDetailsType AS OBJECT (  
    total_value NUMBER,  
    total_orders NUMBER  
);  
/
```

Function 1:

Calculate total orders and total orders values for a customer

```
CREATE OR REPLACE FUNCTION calculate_order_details(p_customer_id IN NUMBER)  
RETURN OrderDetailsType  
IS  
    -- Variable to store the total value of orders  
    v_total_value NUMBER := 0;  
    -- Variable to store the total number of orders  
    v_total_orders NUMBER := 0;  
  
    -- Cursor to iterate over orders of the specified customer  
    CURSOR orders_cursor IS  
        SELECT o.order_id  
        FROM Orders o  
        WHERE o.customer_id = p_customer_id;  
  
    -- Variable to hold the order ID fetched from the cursor  
    v_order_id Orders.order_id%TYPE;  
  
    -- Cursor to fetch the total value of tickets for a specific order  
    CURSOR tickets_cursor IS  
        SELECT SUM(t.price) AS total_order_value  
        FROM Order_Items oi  
        JOIN Tickets t ON oi.ticket_id = t.ticket_id  
        WHERE oi.order_id = v_order_id;  
  
    -- Variable to hold the total value of tickets for the current order  
    v_order_value NUMBER;  
BEGIN  
    -- Open the orders cursor  
    OPEN orders_cursor;  
    LOOP  
        -- Fetch each order ID from the cursor  
        FETCH orders_cursor INTO v_order_id;  
        EXIT WHEN orders_cursor%NOTFOUND; -- Exit loop when no more rows are found  
  
        -- Initialize the order value for the current order  
        v_order_value := 0;  
  
        -- Open the tickets cursor for the current order  
        OPEN tickets_cursor;  
        FETCH tickets_cursor INTO v_order_value;  
        CLOSE tickets_cursor;  
  
        -- Add the current order value to the total value  
        v_total_value := v_total_value + v_order_value;  
        -- Increment the total number of orders  
        v_total_orders := v_total_orders + 1;  
    END LOOP;  
    -- Close the orders cursor  
    CLOSE orders_cursor;  
  
    -- Return the total value and total number of orders as an object  
    RETURN OrderDetailsType(v_total_value, v_total_orders);  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        RETURN OrderDetailsType(0, 0);  
    WHEN OTHERS THEN  
        RAISE;  
END calculate_order_details;  
/
```

Add discount field to orders table.

```
ALTER TABLE Orders ADD discount NUMBER(5, 2);
```

Procedure 1:

Receive a customer id and a total orders value and apply 10% discount to all orders if total orders value is higher than 1000.

```
CREATE OR REPLACE PROCEDURE apply_discount_to_high_value_customer (  
    p_customer_id IN Customers.customer_id%TYPE,  
    p_total_value IN NUMBER  
)  
IS  
BEGIN  
    -- Check if the total value exceeds 1000  
    IF p_total_value > 1000 THEN  
        -- Apply 10% discount to all orders of the customer  
        UPDATE Orders  
        SET discount = 10  
        WHERE customer_id = p_customer_id  
        AND discount IS NULL; -- Update only if discount is not already set  
  
        DBMS_OUTPUT.PUT_LINE('Discount applied to orders for customer ID: ' || p_customer_id);  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Total value is less than 1000. No discount applied.');    END IF;  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);  
END apply_discount_to_high_value_customer;  
/
```

Main program:

A program which receive a customer id as an input, calculates it's total orders value and update each order to get 10% discount.

```
DECLARE  
    v_customer_id NUMBER := &customer_id; -- Input parameter for customer_id  
    v_order_details OrderDetailsType;  
BEGIN  
  
    -- Call function to calculate total orders  
    v_order_details := calculate_order_details(v_customer_id);  
    DBMS_OUTPUT.PUT_LINE('Total Value for Customer ' || v_customer_id || ': ' || v_order_details.total_value);  
    DBMS_OUTPUT.PUT_LINE('Total Orders for Customer ' || v_customer_id || ': ' || v_order_details.total_orders);  
  
    -- Apply discount to high-value customers  
    apply_discount_to_high_value_customer(v_customer_id, v_order_details.total_value);  
  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);  
END;  
/
```

Main program execution.

Present a report displaying four categories:

Revenue > 1000

Revenue between 500 to 1000

Revenue between 100 to 500

Low revenue

The function receives an area id and returns the revenue analysis according to this area.

By default, the function returns analysis for all revenues.

```
-- Create a pipelined function to analyze revenue by area
CREATE OR REPLACE FUNCTION analyze_revenue_by_area (
    p_area_id IN NUMBER DEFAULT NULL
)
RETURN RevenueTableType PIPELINED
IS
    -- Declare local variables
    v_revenue_record RevenueRecordType;
BEGIN
    -- Query to fetch revenue data based on area id and categorize it
    FOR rec IN (
        SELECT
            A.ATTRACTION_NAME,
            SUM(T.PRICE),
            (CASE WHEN SUM(T.PRICE) > 1000 THEN SUM(T.PRICE) ELSE 0 END) AS REVENUE_OVER_1000,
            (CASE WHEN SUM(T.PRICE) > 500 AND SUM(T.PRICE) <= 1000 THEN SUM(T.PRICE) ELSE 0 END) AS REVENUE_OVER_500,
            (CASE WHEN SUM(T.PRICE) > 100 AND SUM(T.PRICE) <= 500 THEN SUM(T.PRICE) ELSE 0 END) AS REVENUE_OVER_100,
            (CASE WHEN SUM(T.PRICE) <= 100 THEN SUM(T.PRICE) ELSE 0 END) AS LOW_REVENUE
        FROM ATTRACTIONS A
        INNER JOIN TICKETS T ON A.ATTRACTION_ID = T.ATTRACTION_ID
        INNER JOIN Order_Items oi ON oi.ticket_id = t.ticket_id
        INNER JOIN LOCATIONS L ON A.LOCATION_ID = L.LOCATION_ID
        WHERE (P_AREA_ID IS NULL OR L.AREA_ID = P_AREA_ID)
        GROUP BY A.ATTRACTION_NAME
    ) LOOP
        -- Populate the record type with fetched data
        v_revenue_record := RevenueRecordType(
            rec.ATTRACTION_NAME,
            rec.REVENUE_OVER_1000,
            rec.REVENUE_OVER_500,
            rec.REVENUE_OVER_100,
            rec.LOW_REVENUE
        );

        -- Pipe the record type to the result set
        PIPE ROW(v_revenue_record);
    END LOOP;

    -- Return the pipelined result set
    RETURN;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL; -- Return NULL when no data found
    WHEN OTHERS THEN
        RAISE; -- Raise exception for other errors
END analyze_revenue_by_area;
/
```

Procedure 2:

Calculate total revenue for attraction.

The procedure receives an attraction name and calculates it's revenue.

```

CREATE OR REPLACE PROCEDURE calculate_total_revenue (
    v_attraction_name IN VARCHAR2,
    p_total_revenue OUT NUMBER
)
IS
    v_total_revenue NUMBER := 0;

    CURSOR revenue_cursor IS
        SELECT price
        FROM Tickets t
        INNER JOIN Order_Items oi ON oi.ticket_id = t.ticket_id
        INNER JOIN Attractions A ON A.Attraction_Id = t.attraction_id
        WHERE attraction_name = v_attraction_name;
    v_price NUMBER;
BEGIN
    OPEN revenue_cursor;
    LOOP
        FETCH revenue_cursor INTO v_price;
        EXIT WHEN revenue_cursor%NOTFOUND;
        v_total_revenue := v_total_revenue + v_price;
    END LOOP;
    CLOSE revenue_cursor;

    -- Store the calculated total revenue in the output parameter
    p_total_revenue := v_total_revenue;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_total_revenue := 0;
    WHEN OTHERS THEN
        RAISE;
END calculate_total_revenue;
/

```

Main program:

The program receives an attraction name and area and then print the attraction revenue and a table with all revenues according to this area

```

DECLARE
    TYPE RevenueRecordType IS RECORD (
        attraction_name VARCHAR2(100),
        revenue_over_1000 NUMBER,
        revenue_over_500 NUMBER,
        revenue_over_100 NUMBER,
        low_revenue NUMBER
    );
    v_attraction_name VARCHAR2(100) := '&attraction_name'; -- Example attraction_name
    v_total_revenue NUMBER;

    -- Define a cursor to fetch data from the pipelined function
    CURSOR c_revenue_records IS
        SELECT * FROM TABLE(analyze_revenue_by_area('&area_name'));

    v_revenue_table RevenueRecordType;
BEGIN
    -- Call function to calculate total revenue
    calculate_total_revenue(v_attraction_name, '&area_name', v_total_revenue);
    DBMS_OUTPUT.PUT_LINE('Total Revenue for ' || v_attraction_name || ': ' || v_total_revenue);
    DBMS_OUTPUT.PUT_LINE('');

    -- Open the cursor
    OPEN c_revenue_records;

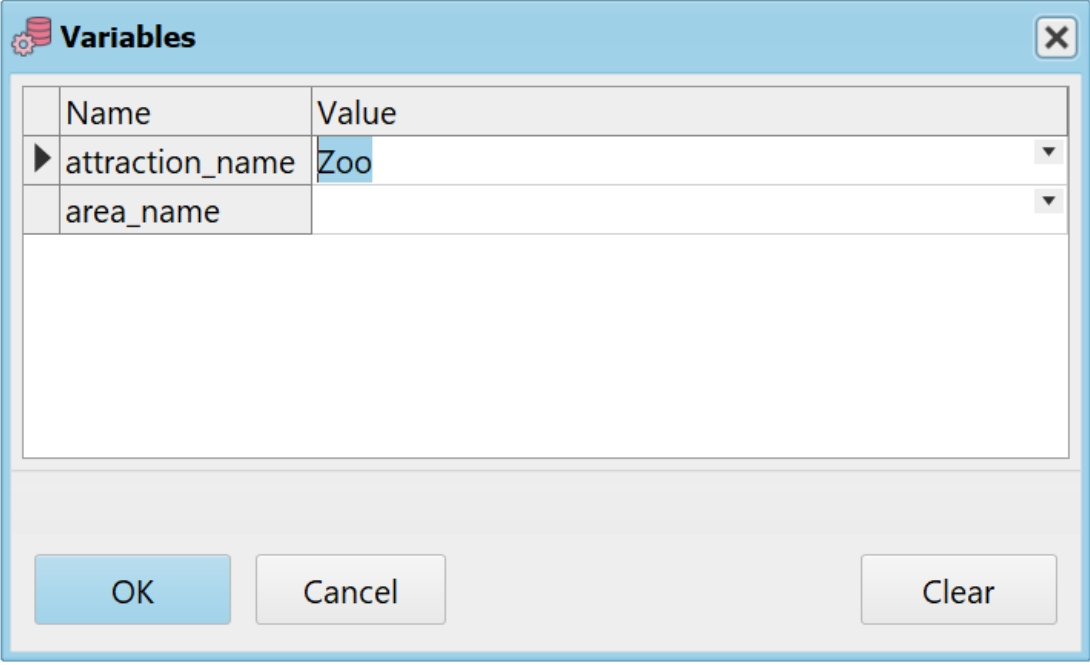
    -- Fetch and process each record
    LOOP
        FETCH c_revenue_records INTO v_revenue_table;
        EXIT WHEN c_revenue_records%NOTFOUND;

        -- Output each row as a table format (you can format as needed)
        DBMS_OUTPUT.PUT_LINE(
            RPAD(v_revenue_table.attraction_name, 20) || ' | ' ||
            RPAD(v_revenue_table.revenue_over_1000, 15) || ' | ' ||
            RPAD(v_revenue_table.revenue_over_500, 15) || ' | ' ||
            RPAD(v_revenue_table.revenue_over_100, 15) || ' | ' ||
            RPAD(v_revenue_table.low_revenue, 15)
        );
    END LOOP;

    -- Close the cursor
    CLOSE c_revenue_records;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/

```

Main program execution:



The image shows a 'Variables' dialog box with a table containing two rows: 'attraction_name' with value 'Zoo' and 'area_name' which is empty. Below the table are 'OK', 'Cancel', and 'Clear' buttons.

Name	Value
attraction_name	Zoo
area_name	

Default – no area selected.

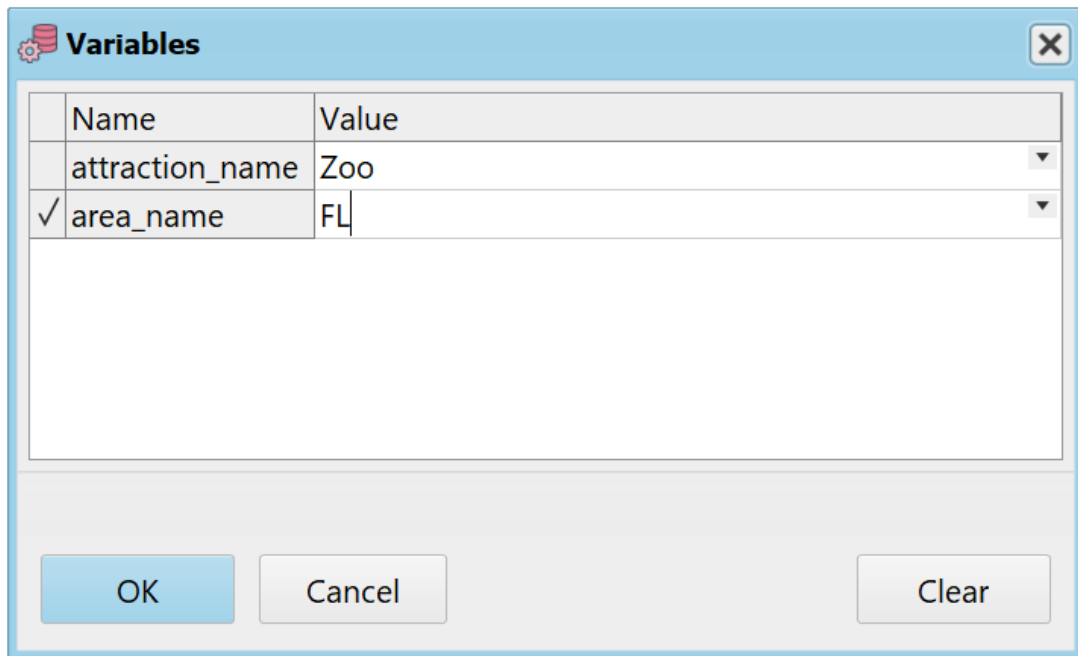
Output:

```

Total Revenue for Zoo: 1419

Famous Bridge      | 1208      | 0          | 0          | 0
Historic Landmark  | 0          | 653        | 0          | 0
Zoo                 | 1419      | 0          | 0          | 0
Observation Deck   | 0          | 792        | 0          | 0
National Park      | 0          | 878        | 0          | 0
Disneyland         | 0          | 952        | 0          | 0
Amusement Park     | 2535      | 0          | 0          | 0
Beach              | 0          | 931        | 0          | 0
Museum             | 0          | 904        | 0          | 0
Public Garden      | 1503      | 0          | 0          | 0
Science Center     | 0          | 889        | 0          | 0
Water Park         | 0          | 0          | 492        | 0
Cultural Festival  | 1354      | 0          | 0          | 0
Universal Studio   | 0          | 820        | 0          | 0
Art Gallery        | 0          | 0          | 425        | 0
Luna Park          | 1022      | 0          | 0          | 0
Botanical Garden   | 0          | 583        | 0          | 0
Aquarium           | 0          | 0          | 210        | 0
Wildlife Sanctuary | 0          | 0          | 331        | 0
Adventure Park     | 0          | 754        | 0          | 0

```



The image shows a 'Variables' dialog box with a table containing two rows. The first row has 'attraction_name' and 'Zoo'. The second row has 'area_name' and 'FL'. The 'area_name' row is selected with a checkmark. Below the table are 'OK', 'Cancel', and 'Clear' buttons.

	Name	Value
	attraction_name	Zoo
✓	area_name	FL

Select FL area.

Output:

Total Revenue for Zoo: 416

Zoo	0	0	416	0
Cultural Festival	0	0	290	0