Experiment no. 9

To create and utilize a **Decision Table** for systematically validating function behavior based on different input scenarios.

Case Study

E-Commerce Discount and Payment Validation System

Aim

To understand and apply **Decision Table Testing** techniques for validating an e-commerce system's functionality involving discount rules and payment validation conditions.

Objectives

- Identify conditions (inputs) and actions (outputs) using domain analysis.
- Construct and simplify a decision table covering all input combinations.
- Derive effective **test cases** from the decision table.
- Compare Decision Table Testing with **Equivalence Class Testing** and **Cause-Effect Graphing** techniques.

Theory

Decision Table Testing is a black-box testing technique used to represent and analyze different combinations of inputs and their corresponding outputs. It helps ensure:

- Comprehensive coverage of input conditions.
- Detection of missing or redundant rules.
- Clear visualization of system behavior in complex logical scenarios.

In this case, the system applies discounts and validates payments based on these rules:

1. If cart amount ≥ ₹2000 → apply 10% discount.
2. If customer is a premium member → apply extra 5% discount.
3. Payment accepted only for valid modes (Credit/Debit/UPI/NetBanking).
4. If payment fails → order is cancelled.
5. Maximum discount ≤ 20% of cart value.

Comparison between Decision Table Testing and Equivalence Class Testing:
Decision Table Testing focuses on identifying and testing **different combinations of input conditions** and their corresponding actions. It is best suited for systems where multiple logical rules interact, such as discount or validation rules.

In contrast, **Equivalence Class Testing (ECT)** divides the input domain into groups or "classes" where each class represents similar behavior. Only one test case from each class is needed, reducing the total number of tests.

While ECT emphasizes **data partitioning** and efficiency, Decision Table Testing emphasizes **complete logical rule coverage** and helps detect missing or conflicting business rules

Procedure

1. **Domain Analysis:**
    - Identify all conditions (inputs) and actions (outputs).
    - Conditions may include: cart amount, membership status, payment validity.
2. **Construct Decision Table:**
    - List all combinations of input conditions.
    - Define corresponding actions (discount, order status, etc.).
3. **Simplify the Table:**
    - Merge redundant or logically equivalent rules.
4. **Derive Test Cases:**
    - Create test cases that ensure full coverage of decision rules.
5. **Compare Techniques:**
    - Discuss advantages of Decision Table Testing over other black-box techniques such as **Equivalence Class Testing** and **Cause-Effect Graphing**.

Program:

```
def process_order(cart_amount, is_premium, payment_valid):
    discount = 0
    order_status = ""
    if cart_amount >= 2000:
        discount += 10
    if is_premium:
        discount += 5
    if discount > 20:
        discount = 20
    if payment_valid:
        order_status = "Accepted"
    else:
        order_status = "Cancelled"
    return {
        "Cart Amount": cart_amount,
        "Premium Member": is_premium,
        "Payment Valid": payment_valid,
        "Final Discount (%)": discount,
        "Order Status": order_status
    }
```

```
test_cases = [
    (1500, False, True),
    (2500, False, True),
    (2500, True, True),
    (4000, True, True),
    (2500, True, False),
    (1500, True, True),
    (1500, False, False),
    (4000, False, True)
]


print("---- Decision Table Test Results ----")
for i, case in enumerate(test_cases, start=1):
    result = process_order(*case)
    print(f"Test Case {i}: {result}")
```

Output

Rule	C1: Cart >= ₹2000	C2: Premium Member	C3: Valid Payment	C4: Payment Success	A1: 10% Disc	A2: +5% Disc	A3: Accept	A4: Cancel
1	F	F	F	F	No	No	No	Yes
2	F	F	F	T	No	No	No	Yes
3	F	F	T	F	No	No	No	Yes
4	F	F	T	T	No	No	Yes	No
5	F	T	F	F	No	No	No	Yes
6	F	T	F	T	No	No	No	Yes
7	F	T	T	F	No	No	No	Yes
8	F	T	T	T	No	Yes	Yes	No
9	T	F	F	F	Yes	No	No	Yes
10	T	F	F	T	Yes	No	No	Yes
11	T	F	T	F	Yes	No	No	Yes
12	T	F	T	T	Yes	No	Yes	No
13	T	T	F	F	Yes	Yes	No	Yes
14	T	T	F	T	Yes	Yes	No	Yes
15	T	T	T	F	Yes	Yes	No	Yes
16	T	T	T	T	Yes	Yes	Yes	No

```
PS C:\Users\VINIT> & C:/Users/VINIT/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/VINIT/Downloads/9.py
---- Decision Table Test Results ----
Test Case 1: {'Cart Amount': 1500, 'Premium Member': False, 'Payment Valid': True, 'Final Discount (%)': 0, 'Order Status': 'Accepted'}
Test Case 2: {'Cart Amount': 2500, 'Premium Member': False, 'Payment Valid': True, 'Final Discount (%)': 10, 'Order Status': 'Accepted'}
Test Case 3: {'Cart Amount': 2500, 'Premium Member': True, 'Payment Valid': True, 'Final Discount (%)': 15, 'Order Status': 'Accepted'}
Test Case 4: {'Cart Amount': 4000, 'Premium Member': True, 'Payment Valid': True, 'Final Discount (%)': 15, 'Order Status': 'Accepted'}
Test Case 5: {'Cart Amount': 2500, 'Premium Member': True, 'Payment Valid': False, 'Final Discount (%)': 15, 'Order Status': 'Cancelled'}
Test Case 6: {'Cart Amount': 1500, 'Premium Member': True, 'Payment Valid': True, 'Final Discount (%)': 5, 'Order Status': 'Accepted'}
Test Case 7: {'Cart Amount': 1500, 'Premium Member': False, 'Payment Valid': False, 'Final Discount (%)': 0, 'Order Status': 'Cancelled'}
Test Case 8: {'Cart Amount': 4000, 'Premium Member': False, 'Payment Valid': True, 'Final Discount (%)': 10, 'Order Status': 'Accepted'}
PS C:\Users\VINIT>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER                    Python

PS C:\Users\VINIT> & C:/Users/VINIT/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/VINIT/Downloads/E5.py
---- Simplified Decision Table Test Results ----
Test Case 1: {'Cart Amount': 1500, 'Premium Member': False, 'Payment Valid': True, 'Final Discount (%)': 0, 'Order Status': 'Accepted'}
Test Case 2: {'Cart Amount': 2500, 'Premium Member': False, 'Payment Valid': True, 'Final Discount (%)': 10, 'Order Status': 'Accepted'}
Test Case 3: {'Cart Amount': 2500, 'Premium Member': True, 'Payment Valid': True, 'Final Discount (%)': 15, 'Order Status': 'Accepted'}
Test Case 4: {'Cart Amount': 1500, 'Premium Member': True, 'Payment Valid': True, 'Final Discount (%)': 5, 'Order Status': 'Accepted'}
Test Case 5: {'Cart Amount': 3000, 'Premium Member': True, 'Payment Valid': False, 'Final Discount (%)': 0, 'Order Status': 'Cancelled'}
PS C:\Users\VINIT>
```

Conclusion

Through this experiment, the Decision Table Testing technique was successfully applied to validate the functional behavior of an e-commerce discount and payment validation system. By systematically identifying all possible input conditions and their corresponding actions, the decision table helped ensure comprehensive test coverage and logical consistency. The simplified rules enabled efficient test case generation and reduced redundancy in validation. Compared to Equivalence Class Testing and Cause-Effect Graphing, Decision Table Testing provided a structured and visual approach for handling complex condition combinations, making it highly effective for verifying rule-based business logic.