We know that ***Black box testing*** involves validating the system without knowing its internal design. Moreover, we have also discussed that ***Boundary Value Analysis*** and ***Equivalence Partitioning*** can only handle one input condition at a time. So what do we do if we need to test complex business logic where multiple input conditions and actions are involved? Consequently, we will discuss another black box testing technique known as ***Decision Table Testing***. Because this testing can handle such cases.

To get a better understanding, it's recommended that you read our articles on ***Boundary Value Analysis*** and ***Equivalence Partitioning***before proceeding further. Subsequently, we will cover the below items in this article.

What is Decision Table Testing?
How to create a Decision Table?
Pitfalls of Decision Table Testing

What is Decision Table Testing?

We often test applications where the action is basis of multiple input combinations. *E.g., if you are applying for a loan, then the eligibility for getting a loan will depend on several factors like age, Monthly Salary, loan amount required, etc.* Consequently, these combinations are called ***business logic*** based on which the application functionality works. The decision table is a black box testing technique that is used to test these complex business logic. It's a tabular representation of input conditions and resulting actions. Additionally, it shows the causes and effects. Therefore, this technique is also called a ***cause-effect table.***

CONDITIONS	***1st Input***	***2nd Input***	***3rd Input***
Salaried?	*Y*	*N*	*Y*
Monthly Income > 25000	*N*	*Y*	*Y*
IT Returns available?	*Y*	*N*	*Y*
ACTIONS			
Loan Eligibility	*N*	*N*	*Y*

*Therefore, it is a simple decision table where **Salaried, Monthly Income,** and **IT Returns** are input conditions. In addition to the above, their combination results in the final action (Loan Eligibility).*

Testing combinations can be a challenge, especially if the number of combinations is enormous. Moreover, testing all combinations is not practically feasible as it's not cost and time effective. Therefore, we have to be satisfied with testing just a small subset of combinations. That is to say, the success of this technique depends on our choice of combinations

Consequently, let's dive into practical examples and see the methods and best practices to create decision tables.

How to Create a Decision Table?

The decision table works on input conditions and actions. That is to say, we create a Table in which the top rows are input conditions, and in the same vein, the bottom rows are resulting actions. Similarly, the columns correspond to unique combinations of these conditions.

Common Notations for Decision Table

- ***For Conditions***
  - *Y means the condition is True. We can depict it as **T** Or **1.***
  - *N indicates the condition is False. We can also refer to it as **F** Or **0.***
  - ***-** It means the condition doesn't matter. We can represent it as **N/A.***
- ***For Actions***
  - ***X** means action should occur. We can also refer to it as **Y** Or **T** Or **1.***
  - *Blank means action should not happen. We can also represent it as **N** Or **F** Or **0.***

Now that we are clear on notations, let's consider the below scenario.

Consider a Banking application that will ask the user to fill a personal loan application online. Based on the inputs, the application will display real-time whether the loan will get approval, rejection, or requires a visit to the branch for further documentation and discussion. Let's assume that the loan amount is 5L. In addition to this, we won't change it to reduce the complexity of this scenario.

Accordingly, the application has the following business rules:

- *If you are Salaried and your Monthly Salary is greater than or equal to 75k, then your loan will be approved.*
- *If you are Salaried and your Monthly Salary is between 25k and 75k, then you will need to visit the branch for further discussion.*
- *If you are Salaried and your Monthly Salary is less than 25k, then your loan will be rejected.*
- *If you are Not Salaried and your Monthly Salary is greater than or equal to 75k, then you will need to visit the branch for further discussion.*
- *If you are Not Salaried and your Monthly Salary is less than 75k, then your loan will be rejected, consequently.*

There are four steps that we follow to create a Decision table. Subsequently, let's have a look at these steps based on the above scenario.

Step 1 - Identify all possible Conditions

Consequently, the possible conditions are as below:

- *First, whether the person is Salaried or not.*

Second, Monthly Salary of the applicant.

Step 2 - Identify the corresponding actions that may occur in the system

Therefore, the possible actions are:

Should the loan be approved (Possible values Y or N)
Should the loan be rejected (Possible values Y or N)
Should the applicant be called for further documentation and discussion (Possible values Y or N)

Step 3 - Generate All possible Combinations of Conditions

Each of these combinations forms a column of the decision table. Firstly, let's see how many variations are possible for each condition:

Condition 1 - *Whether the person is salaried or not - 2 Variations: Y or N*
Condition 2 - *Monthly Salary of the Applicant - 3 Variations : <25k , 25k - 75k and >75k*

Subsequently, based on this our total combinations will come up as 2*3 = 6

1st Combination : *Salaried = Yes , Monthly Salary < 25k*
2nd Combination: *Salaried = Yes, Monthly Salary 25k - 75k*
3rd Combination: *Salaried = Yes, Monthly Salary >75k*
4th Combination : *Salaried = No , Monthly Salary < 25k*
5th Combination: *Salaried = No, Monthly Salary 25k - 75k*
6th Combination: *Salaried = No, Monthly Salary >75k*

At this point, our Decision table will look like below:

CONDITIONS	***Combination # 1***	***Combination # 2***	***Combination # 3***	***Combination # 4***	***Combination # 5***	***Combination # 6***
Salaried?	*Y*	*Y*	*Y*	*N*	*N*	*N*
Monthly Income < 25k	*Y*	*NA*	*NA*	*Y*	*NA*	*NA*
Monthly Income 25k - 75k	*NA*	*Y*	*NA*	*NA*	*Y*	*NA*
Monthly Income >75k	*NA*	*NA*	*Y*	*NA*	*NA*	*Y*
ACTIONS						
Loan Approved						
Loan Rejected						
Further Documentation & Visit						

Step 4 - Identify Actions based on the combination of conditions

Subsequently, the next step will be identifying actions for each combination based on the business logic as defined for the system.

***1st Combination** - Action = Loan Rejected*
***2nd Combination** - Action = Further Documentation & Visit*
***3rd Combination** - Action = Loan Approved*
***4th Combination** - Action = Loan Rejected*
***5th Combination** - Action = Loan Rejected*
***6th Combination** - Action = Further Documentation & Visit*

After this, our final decision table will look like below:

CONDITIONS	***Combination # 1***	***Combination # 2***	***Combination # 3***	***Combination # 4***	***Combination # 5***	***Combination # 6***
Salaried?	*Y*	*Y*	*Y*	*N*	*N*	*N*
Monthly Income < 25k	*Y*	*NA*	*NA*	*Y*	*NA*	*NA*
Monthly Income 25k - 75k	*NA*	*Y*	*NA*	*NA*	*Y*	*NA*
Monthly Income >75k	*NA*	*NA*	*Y*	*NA*	*NA*	*Y*
ACTIONS						
Loan Approved	*NA*	*NA*	*Y*	*NA*	*NA*	*NA*
Loan Rejected	*Y*	*NA*	*NA*	*Y*	*Y*	*NA*
Further Documentation & Visit	*NA*	*Y*	*NA*	*NA*	*NA*	*Y*

Now that we have created the decision table, what's next? Consequently, we have to use this decision table to generate test cases for each of these combinations. The actions will be our expected output. Subsequently, we will compare this to the actual results to pass or fail the ***test cases***.

In addition to the above, the typical minimum coverage standard for decision table testing is to have at least one test case per decision rule in the table. That is to say; it generally involves covering all combinations of conditions. In addition to that, we measure the coverage as the number of decision rules tested by at least one test case, divided by the total number of decision rules. Consequently, we express it as a percentage.

Pitfalls:

To conclude, we have seen that Decision table testing is immensely beneficial when we have to test complex business rules. However, like every other technique, there are pitfalls for the decision table as well that we should know.

***Capturing All Combinations and Actions - Firstly,** it's often challenging to ensure that we have got all the input combinations and actions captured. In other words, there are several combinations in large systems that it's easy to miss out on them. In addition to that, we should*

remember that the success of the decision table solely relies on our ability to capture these combinations correctly.
Redundant Combinations - Secondly, *the decision table could also introduce redundant combinations if we solely go by the permutations. E.g., If the number of conditions is six and each condition can take three values. As a result, our total combination will become 333333* = 729 !! Do you think we can cover all these combinations in our test cases? It's practically impossible unless there is infinite time & money! So what do we do in such cases? The bare minimum requirement for any decision table is that the variation of each condition is covered at least once. Therefore, if you ignore all the permutations, there are six conditions with three variations each. Which, in turn, means that 18 combinations (6*3) can cover each of these variations at-least once. Moreover, there is also a possibility that your one combination can cover multiple conditions. As a result, we can further reduce the combinations. Additionally, while selecting these combinations, we need to ensure that the selection of all our actions happens at least once.*

For instance, if you look at our earlier example, do you realize that combination 4 and 5 are redundant? In other words, there is no need to keep both. More importantly, the business rule says Salary less than 75k for the non-salaried person, and it doesn't need to split further!

Boundary and Edge Conditions - Lastly, *let's assume we have taken care of all the right things and pitfalls. In our earlier example, one of the test conditions is to have Salary <25k, and another where Salary is between 25k - 75k. Merely applying the decision table, you can take the input value as 20k and 50k. Which, in turn, will suffice the requirement of the decision table. However, if you remember, we get more errors in boundary conditions. Which, in fact, are entirely missing here! In addition to that, as there is a range involved here, we can use Equivalence partitioning as well, along with Boundary value. Please read our article on Boundary Value Analysis if you want to know why we will get more errors in boundary conditions. Additionally, you can also understand how to apply Equivalence partitioning along with boundary value.*

Moreover, the decision table alone is not sufficient to catch all the defects. Therefore, we should always use it with boundary value and Equivalence partitioning techniques wherever possible.

In our earlier example, the input condition for Salary can be taken as 24999, 25000, and 25001 for the 25k input.

However, what happens when the system behavior changes even when you give the same input? *E.g., If you are entering a wrong password multiple times, you will get a message that your password is incorrect. However, once your user id gets locked, you will get a different message that you cannot access your account anymore.* Conclusively, all this happens even when the input condition remains the same. Therefore, you would have figured out by now that we cannot handle this situation using the Decision table. Hence, we will need to learn another technique called ***State Transition testing*** that will cover such conditions.