Pairwise Testing Example

Car Ordering Application:

- The car ordering application allows for Buying and Selling cars. It should support trading in Delhi and Mumbai.
- The application should have registration numbers, may be valid or invalid. It should allow the trade of following cars: BMW, Audi, and Mercedes.
- Two types of booking can be done: E-booking and In Store.
- Orders can be placed only during trading hours.

<u>Step #1</u>: Let's list down the variables involved.
1) Order category
a. Buy
b. Sell
2) Location
a. Delhi
b. Mumbai
3) Car brand
a. BMW
b. Audi
c. Mercedes
4) Registration numbers
a. Valid (5000)
b. Invalid
5) Order type
a. E-Booking
b. In-store
6) Order time
a. Working hours
b. Non-working hours

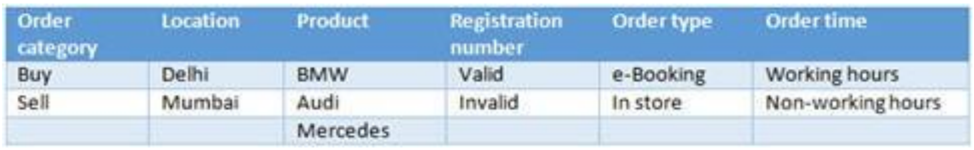
If we want to test all possible valid combinations:
= 2 X 2 X 3 X 5000 X 2 X 2
= 240000 Valid test cases combinations :(
There is also an infinite number of invalid combinations.

<u>Step #2</u>: Let's simplify
– Use a smart representative sample.
– Use groups and boundaries, even when data is non-discrete.
– Reduce Registration Number to Two

1. Valid registration number
2. Invalid registration number

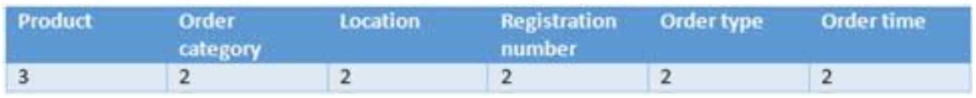
Now let's calculate the number of possible combinations
= 2 X 2 X 3 X 2 X 2 X 2
= 96

<u>Step #3</u>: Arranging variables and values involved.
When we arrange variables and values involved, it looks something like this.



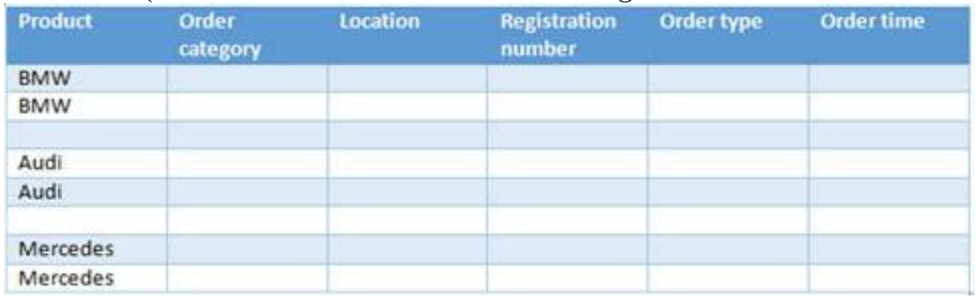
Order category	Location	Product	Registration number	Order type	Order time
Buy	Delhi	BMW	Valid	e-Booking	Working hours
Sell	Mumbai	Audi	Invalid	In store	Non-working hours
		Mercedes			

Now order the variables so that the one with the most number of values is first and the least is last.



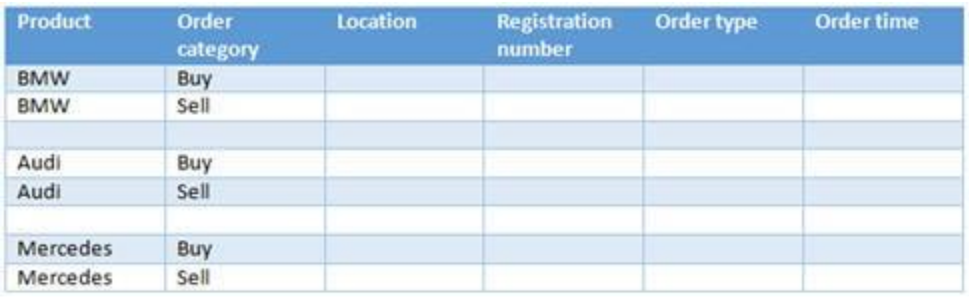
Product	Order category	Location	Registration number	Order type	Order time
3	2	2	2	2	2

<u>Step #4</u>: Arrange variables to create a test suite
Let's start filling in the table column by column. Initially, the table should look something like this. The three values of **Product** (variable having the highest number of values) should be written two times each (two is the number of values of next highest variable i.e. **Order category**).



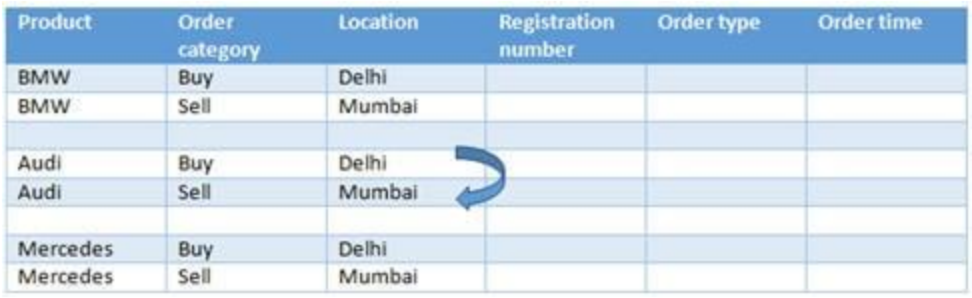
Product	Order category	Location	Registration number	Order type	Order time
BMW					
BMW					
Audi					
Audi					
Mercedes					
Mercedes					

The Order Category column has two values. That's how many times we need to insert the values of the first column, Product.



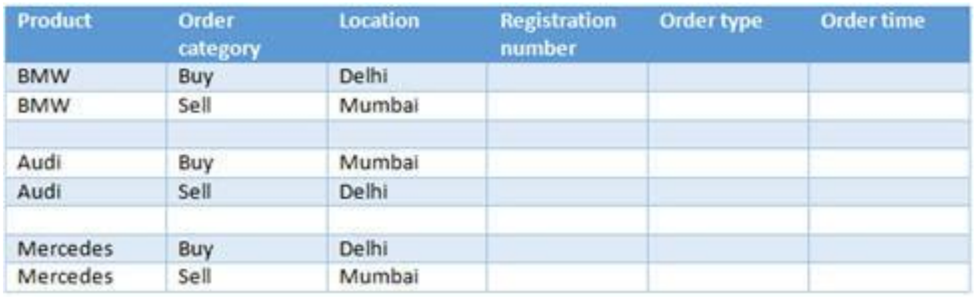
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy				
BMW	Sell				
Audi	Buy				
Audi	Sell				
Mercedes	Buy				
Mercedes	Sell				

For each set of values in column 1, we put both values of column 2. Repeat the same for column 3.



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Delhi			
Audi	Sell	Mumbai			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

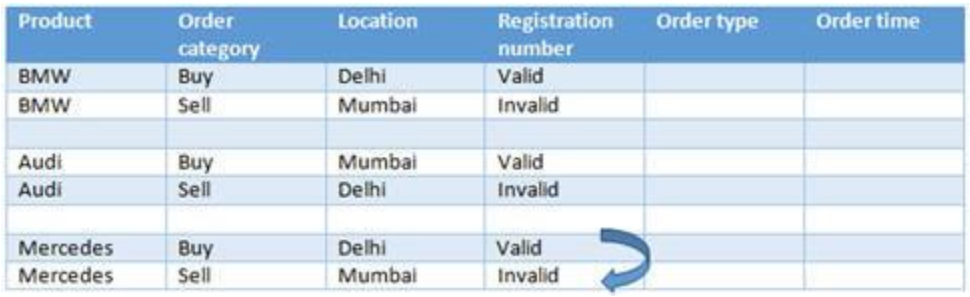
We have a Buy and Delhi, but wait – there’s no Buy and Mumbai. We have a Sell and Mumbai, but there’s no Sell and Delhi. Let’s swap around the values in the second set in the third column.



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Mumbai			
Audi	Sell	Delhi			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

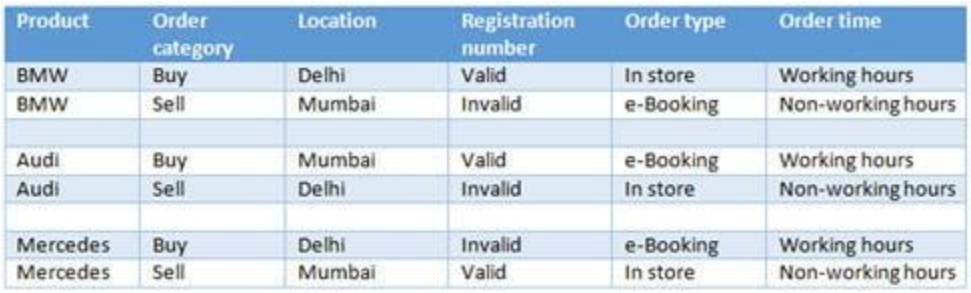
This looks much better!

We will repeat the same steps for column 3 and 4.



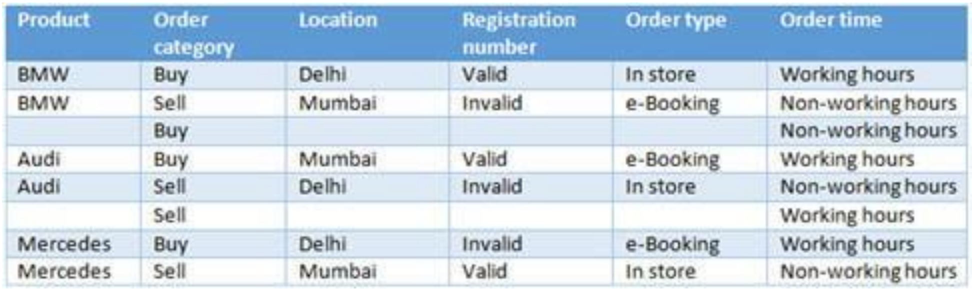
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid		
BMW	Sell	Mumbai	Invalid		
Audi	Buy	Mumbai	Valid		
Audi	Sell	Delhi	Invalid		
Mercedes	Buy	Delhi	Valid		
Mercedes	Sell	Mumbai	Invalid		

When columns 3 and 4 are compared, each value in column 3 has both the values of column 4. But when you compare the 2nd and 4th column, we have Buy and Valid & Sell and Invalid .i.e. Buy does not have ‘Invalid’ and Sell does not have ‘Valid’. Hence we need to interchange the last set of values in the 4th column.



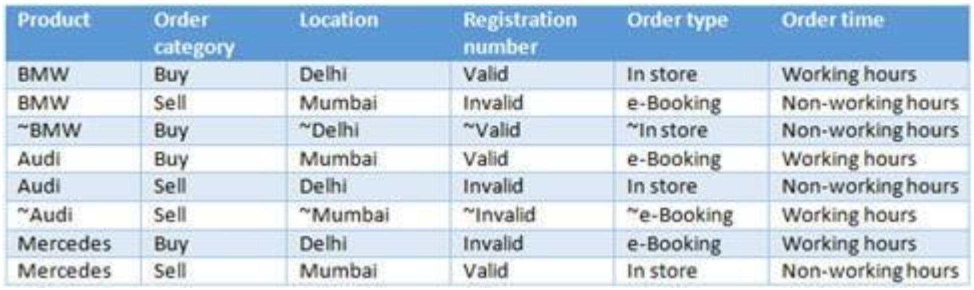
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

Column 6 (Order time) is problematic. We are missing Buy/Non-working hours and Sell/Working hours. We can’t fit our missing pairs by swapping around values as we already swapped all the rows if we swap now we may miss other possible pairs which are already sorted. So, we add two more test cases that contain these pairs. Hence, the blank rows!



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
	Buy				Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
	Sell				Working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

Now we will fill in the empty cells as we desire because the other variable values are purely arbitrary (or Don't Cares ~).



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
~BMW	Buy	~Delhi	~Valid	~In store	Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
~Audi	Sell	~Mumbai	~Invalid	~e-Booking	Working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

Hurray! All pairs in 8 cases, instead of all combinations in 96!

Hence, we saw how efficient All-pairs technique of test design is. There stands a good chance of finding bugs and it is fun and powerful.
The pairwise testing technique has some limitations as well.

- It fails when the values selected for testing are incorrect.
- It fails when highly probable combinations get too little attention.
- It fails when interactions between the variables are not understood well.

Pairwise Testing Tools:

Tools are available that applies the all-pairs testing technique that facilitates us to effectively automate the Test Case Design process by generating a compact set of parameter value choices as the desired Test Cases. Some well-known tools from the industry are:

- PICT – 'Pairwise Independent Combinatorial Testing', provided by Microsoft Corp.
- IBM FoCuS – 'Functional Coverage Unified Solution', provided by IBM.
- ACTS – 'Advanced Combinatorial Testing System', provided by NIST, an agency of the US Government.
- Hexawise
- Jenny
- Pairwise by Inductive AS
- VPTag free All-Pair Testing Tool