```
MariaDB [SalesOrdersExample]> SELECT * FROM Products WHERE ProductNumber=1;
+---------------+-------------------------+--------------------+-------------+----------------+------------+
| ProductNumber | ProductName             | ProductDescription | RetailPrice | QuantityOnHand | CategoryID |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
|             1 | Trek 9000 Mountain Bike | NULL               |     1200.00 |              4 |          2 |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
1 row in set (0.015 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Orders WHERE OrderNumber=945;
+-------------+------------+------------+------------+------------+
| OrderNumber | OrderDate  | ShipDate   | CustomerID | EmployeeID |
+-------------+------------+------------+------------+------------+
|         945 | 2015-09-04 | 2015-09-05 |       1004 |        701 |
+-------------+------------+------------+------------+------------+
1 row in set (0.014 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Order_Details WHERE OrderNumber=945;
+-------------+---------------+-------------+-----------------+
| OrderNumber | ProductNumber | QuotedPrice | QuantityOrdered |
+-------------+---------------+-------------+-----------------+
|         945 |             1 |     1200.00 |               2 |
+-------------+---------------+-------------+-----------------+
1 row in set (0.010 sec)
```

Here is what happened after running first command of T2 and running all commands in T1. As we can see, there is no change yet, since first command in T2 is not a transaction yet.(Assume that OrderNumber 945 has existed, we are going to create a new entry).

```
MariaDB [SalesOrdersExample]> INSERT INTO Orders (OrderNumber, OrderDate, ShipDate, CustomerID, EmployeeID) VALUES (946, '2015-09-04', '2015-09-05', 1004, 701);
Query OK, 1 row affected (0.010 sec)

MariaDB [SalesOrdersExample]> INSERT INTO Order_Details (OrderNumber, ProductNumber, QuotedPrice, QuantityOrdered) VALUES (946, 1, 1200.00, 2);
Query OK, 1 row affected (0.003 sec)
```

```
MariaDB [SalesOrdersExample]> SELECT * FROM Products WHERE ProductNumber=1;
+---------------+-------------------------+--------------------+-------------+----------------+------------+
| ProductNumber | ProductName             | ProductDescription | RetailPrice | QuantityOnHand | CategoryID |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
|             1 | Trek 9000 Mountain Bike | NULL               |     1200.00 |              4 |          2 |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
1 row in set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Orders WHERE OrderNumber=946;
Empty set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Order_Details WHERE OrderNumber=946;
Empty set (0.000 sec)
```

After running both remaining statements of T2, here is what happened when we run T1 in left workbench. We have not committed yet, that's why there is still empty set.

```
MariaDB [SalesOrdersExample]> SELECT * FROM Products WHERE ProductNumber = 1;
+---------------+-------------------------+--------------------+-------------+----------------+------------+
| ProductNumber | ProductName             | ProductDescription | RetailPrice | QuantityOnHand | CategoryID |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
|             1 | Trek 9000 Mountain Bike | NULL               |     1200.00 |              2 |          2 |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
1 row in set (0.001 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Orders WHERE OrderNumber = 946;
+-------------+------------+------------+------------+------------+
| OrderNumber | OrderDate  | ShipDate   | CustomerID | EmployeeID |
+-------------+------------+------------+------------+------------+
|         946 | 2015-09-04 | 2015-09-05 |       1004 |        701 |
+-------------+------------+------------+------------+------------+
1 row in set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Order_Details WHERE OrderNumber = 946;
+-------------+---------------+-------------+-----------------+
| OrderNumber | ProductNumber | QuotedPrice | QuantityOrdered |
+-------------+---------------+-------------+-----------------+
|         946 |             1 |     1200.00 |               2 |
+-------------+---------------+-------------+-----------------+
1 row in set (0.000 sec)
```

After running T1 in the right workbench, we could see that data in this workbench has updated to the new one that we entered earlier.

```
MariaDB [SalesOrdersExample]> SELECT * FROM Products WHERE ProductNumber=1;
+---------------+-------------------------+--------------------+-------------+----------------+------------+
| ProductNumber | ProductName             | ProductDescription | RetailPrice | QuantityOnHand | CategoryID |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
|             1 | Trek 9000 Mountain Bike | NULL               |     1200.00 |              4 |          2 |
+---------------+-------------------------+--------------------+-------------+----------------+------------+
1 row in set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Orders WHERE OrderNumber=946;
Empty set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Order_Details WHERE OrderNumber=946;
Empty set (0.001 sec)
```

When we rerun again T1 in the left after committing T2 in the right workbench, we could see that T1 still has empty set.

```
MariaDB [SalesOrdersExample]> COMMIT;
Query OK, 0 rows affected (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Orders WHERE OrderNumber=946;
+-------------+------------+------------+------------+------------+
| OrderNumber | OrderDate  | ShipDate   | CustomerID | EmployeeID |
+-------------+------------+------------+------------+------------+
|         946 | 2015-09-04 | 2015-09-05 |       1004 |        701 |
+-------------+------------+------------+------------+------------+
1 row in set (0.000 sec)

MariaDB [SalesOrdersExample]> SELECT * FROM Order_Details WHERE OrderNumber=946;
+-------------+---------------+-------------+-----------------+
| OrderNumber | ProductNumber | QuotedPrice | QuantityOrdered |
+-------------+---------------+-------------+-----------------+
|         946 |             1 |     1200.00 |               2 |
+-------------+---------------+-------------+-----------------+
1 row in set (0.000 sec)

MariaDB [SalesOrdersExample]>
```

After committing T1 in left workbench, we could see that finally, data in T1 has been updated and displayed.

Responses:

-As shown here, the isolation level that we are working is repeatable read.

```
MariaDB [SalesOrdersExample]> show variables like "tx_isolation";
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+
1 row in set (0.001 sec)
```

-The transaction could see the changes made for making changes in the same workbench that it was entered. For transaction that reads changes, they could only be readed by the time it was committed in each workbench that it was entered.

- T1 cannot see the result of T2 after T2 being commited because for repeatable-read level, it could only see the commited results of other transactions before the transaction begins. So we can only see results of transaction before T1 and not when it's happening.

-Repeatable read is an isolation level in MySQL. It means that we could only see value given by a different item committed before this transaction. Phantoms are new rows that are appeared during a read transaction, because of another write transaction in a different workbench (in this case, entering remaining commands of T2 is creating phantoms).

-In MySQL standard, Repeatable-read is the isolation level that permit transactions to read data representation of the first reading time or an item committed before it.But it still allows for new data inserted in another workbench to pop up (or phantom), so in contrast to serializable, repeatable-read could not eliminate phantom.