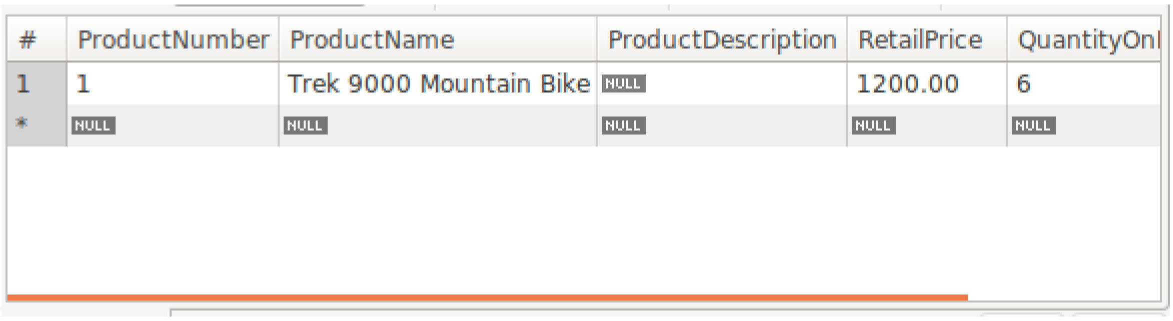
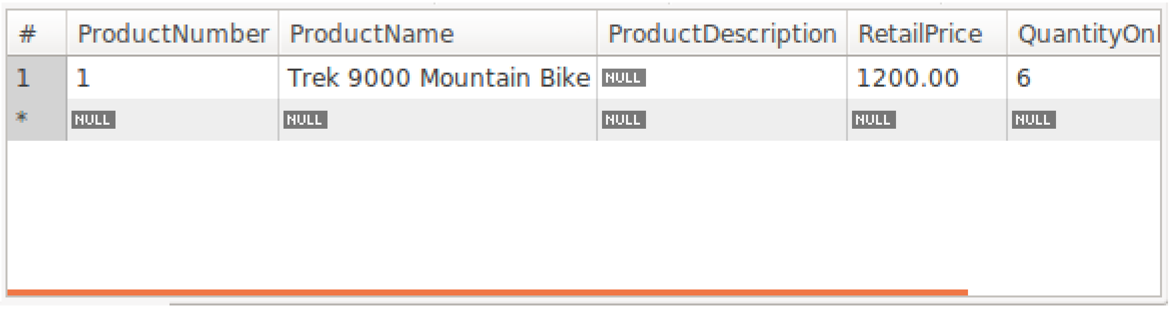


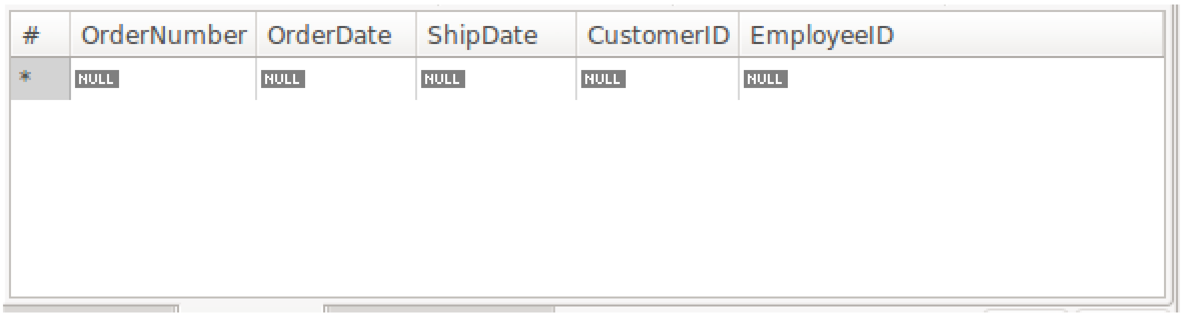
#	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnl
1	1	Trek 9000 Mountain Bike	NULL	1200.00	6
*	NULL	NULL	NULL	NULL	NULL

T1 starts with 6 quantities

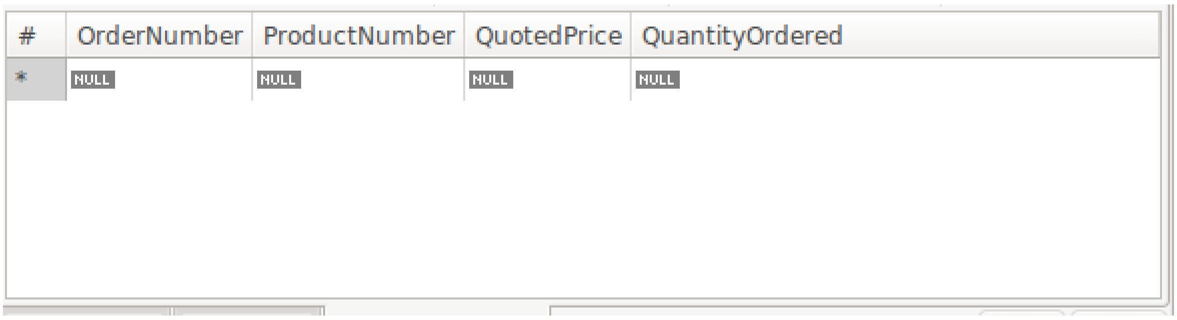


#	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnl
1	1	Trek 9000 Mountain Bike	NULL	1200.00	6
*	NULL	NULL	NULL	NULL	NULL

After running the UPDATE statement on T2, T1 still has 6 quantities.

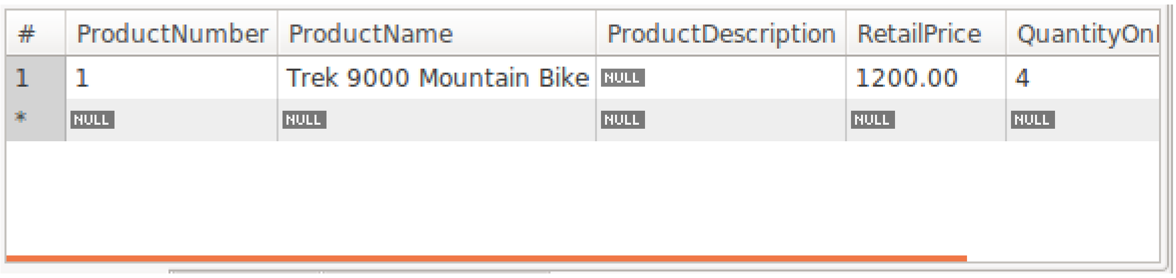


#	OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID
*	NULL	NULL	NULL	NULL	NULL

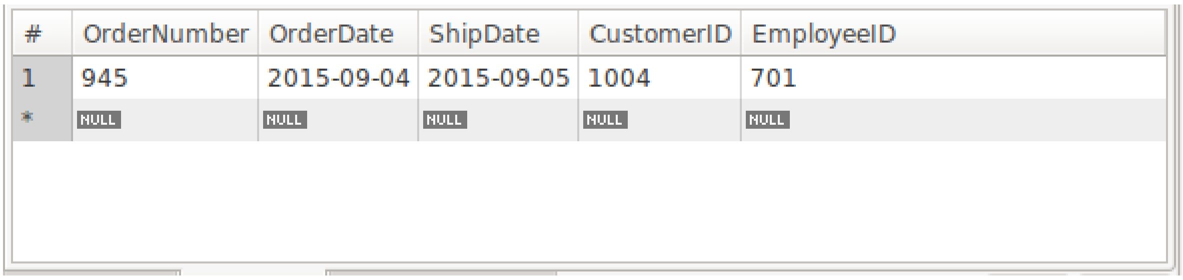


#	OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
*	NULL	NULL	NULL	NULL

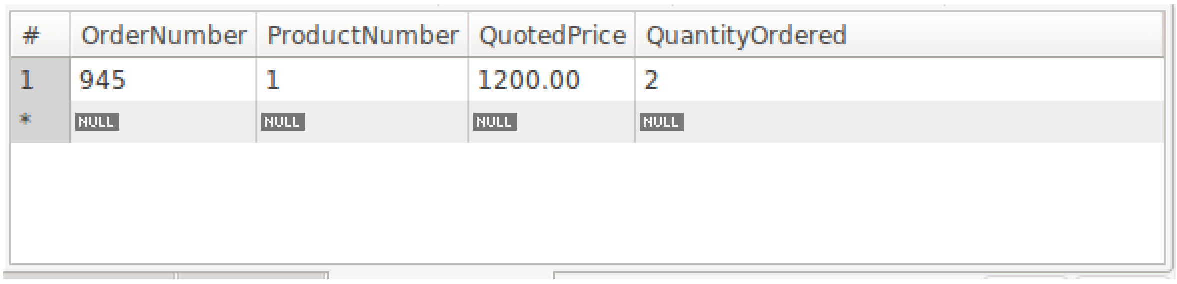
After running the two INSERT statements on T2, T1'S Order and Order_Details table remained empty.



#	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnl
1	1	Trek 9000 Mountain Bike	NULL	1200.00	4
*	NULL	NULL	NULL	NULL	NULL



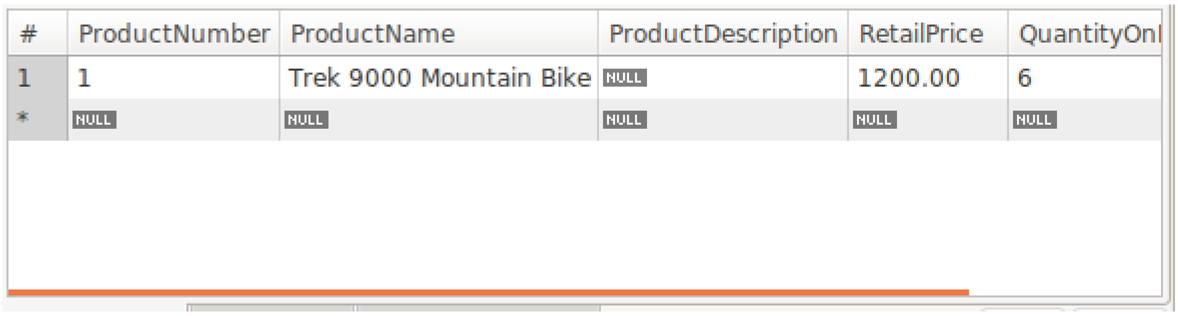
#	OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID
1	945	2015-09-04	2015-09-05	1004	701
*	NULL	NULL	NULL	NULL	NULL



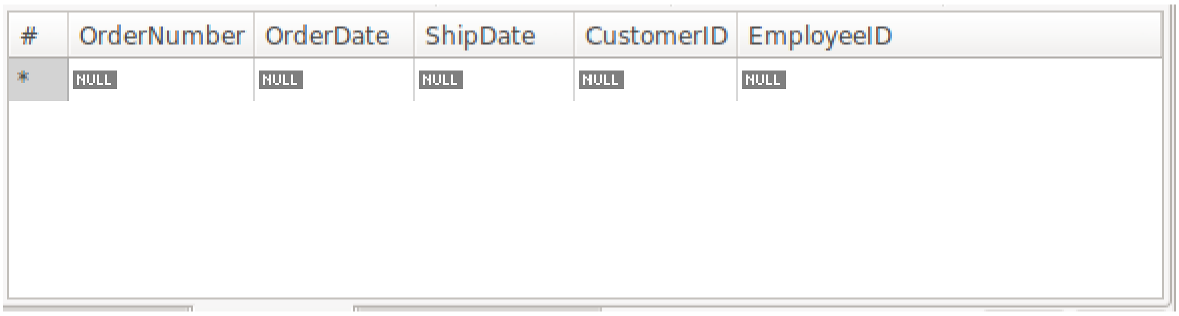
#	OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
1	945	1	1200.00	2
*	NULL	NULL	NULL	NULL

Copying and running three queries in T2 gave a different result. They were affected by the previous statements.

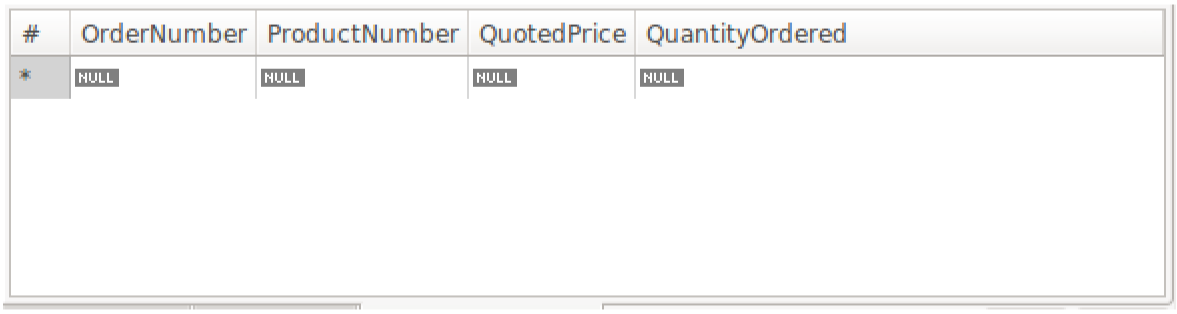
After committing T2



#	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnl
1	1	Trek 9000 Mountain Bike	NULL	1200.00	6
*	NULL	NULL	NULL	NULL	NULL

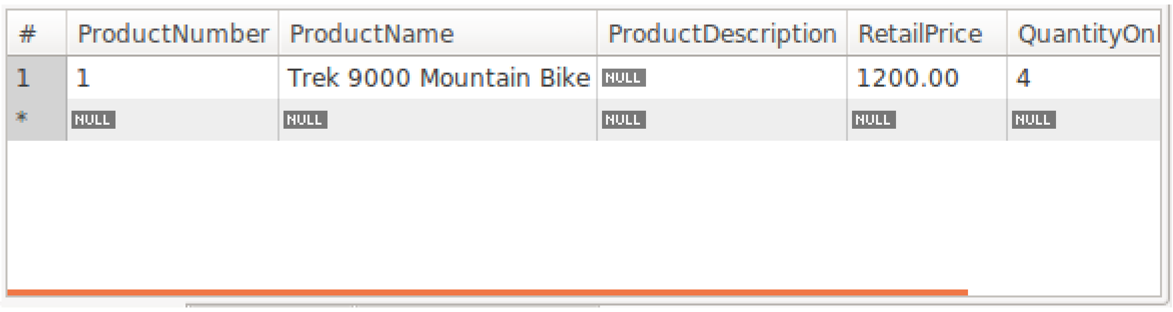


#	OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID
*	NULL	NULL	NULL	NULL	NULL

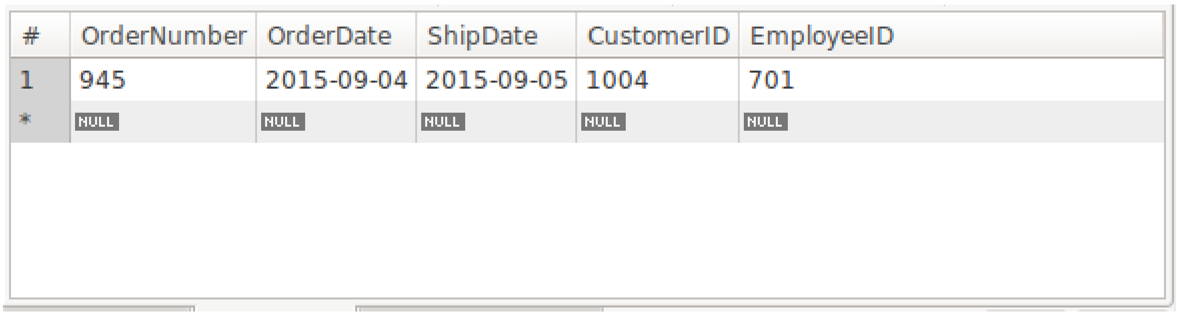


#	OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
*	NULL	NULL	NULL	NULL

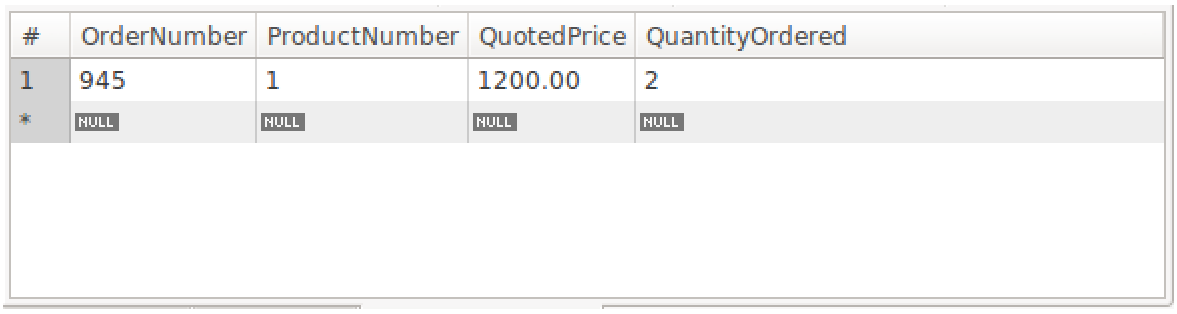
After committing T1



#	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnl
1	1	Trek 9000 Mountain Bike	NULL	1200.00	4
*	NULL	NULL	NULL	NULL	NULL



#	OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID
1	945	2015-09-04	2015-09-05	1004	701
*	NULL	NULL	NULL	NULL	NULL



#	OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
1	945	1	1200.00	2
*	NULL	NULL	NULL	NULL

T1 and T2 are at the repeatable-read isolation level.

A transaction sees the changes after running commit. After running the transaction in T2, followed by the queries statements in T2, we could see that the changes had taken effect. However, running the queries in T1, showed that table was still the same as it was at the beginning. After running commit on T2 and T1, followed by running the queries on T1, the changes went through.

T1 can’t see changes of T2 when T2 commits

The repeatable-read isolation level guarantees that any data read cannot change. After committing T2, when T1 read the same data again, the previously read data was returned, unchanged. After running commit for T1, the data read has updated. We do have phantoms here in MySQL. Phantom read occurs when the same query returns a different result, due to an INSERT or DELETE statement in another transaction.