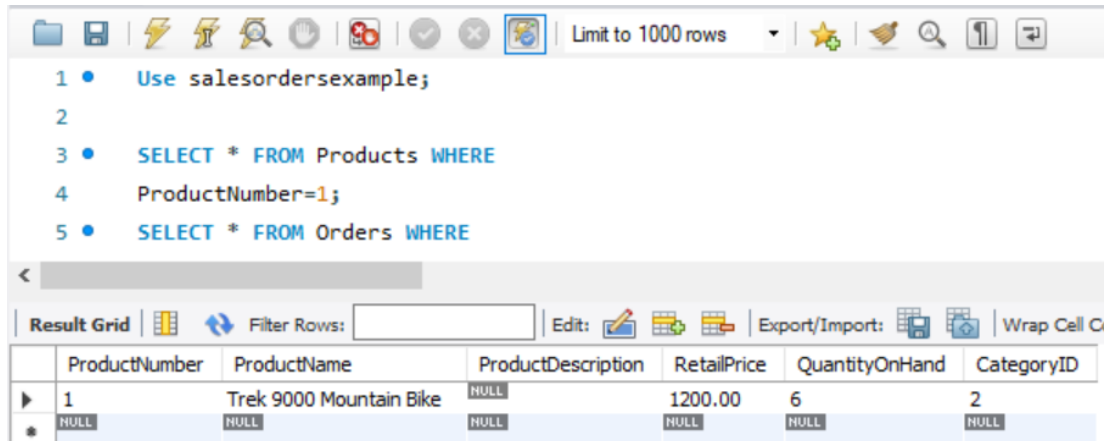


1) It gives null values for 2nd and 3rd query and the 1st query doesn't change the production hand number

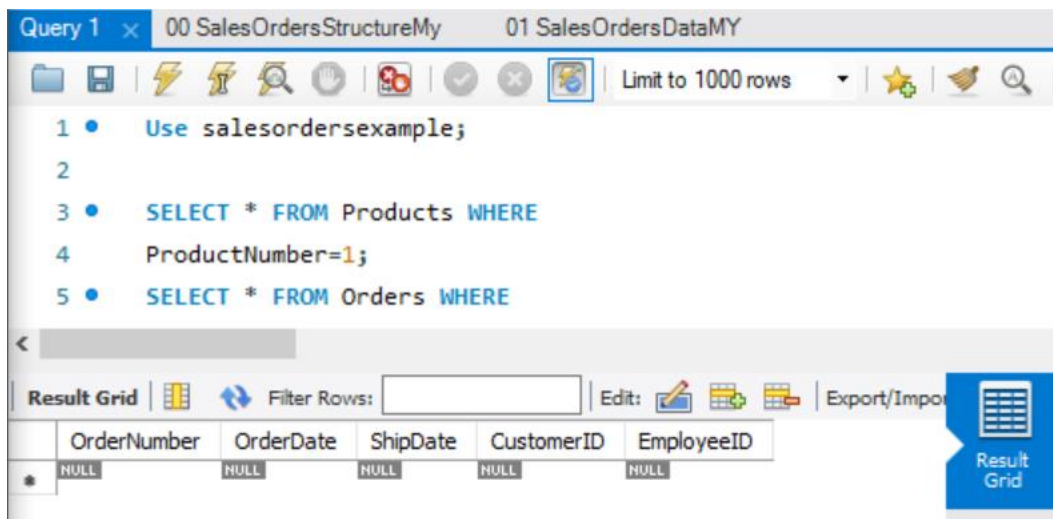


Query window showing the following SQL code:

```
1 • Use salesordersexample;  
2  
3 • SELECT * FROM Products WHERE  
4   ProductNumber=1;  
5 • SELECT * FROM Orders WHERE
```

Result Grid:

ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
1	Trek 9000 Mountain Bike	NULL	1200.00	6	2

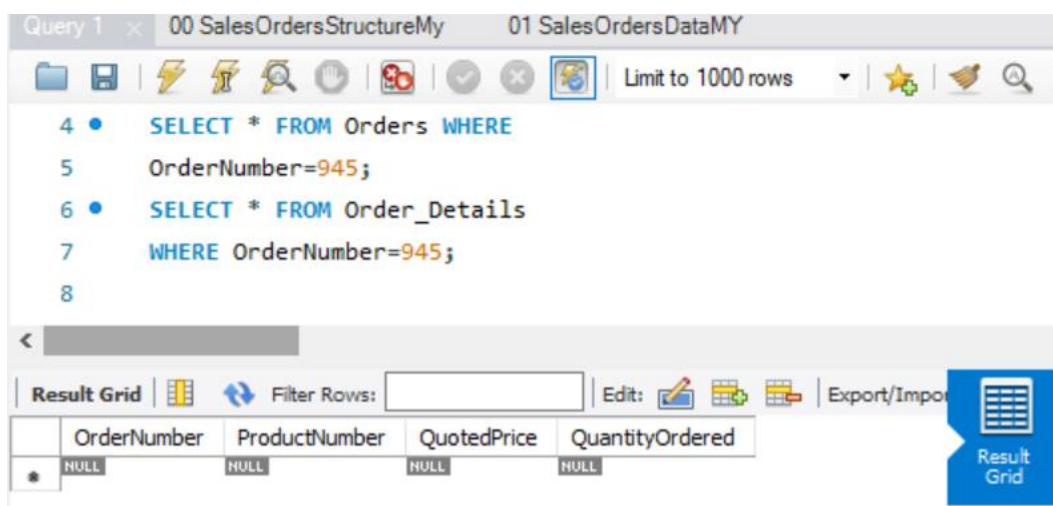


Query window showing the following SQL code:

```
1 • Use salesordersexample;  
2  
3 • SELECT * FROM Products WHERE  
4   ProductNumber=1;  
5 • SELECT * FROM Orders WHERE
```

Result Grid:

OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID
NULL	NULL	NULL	NULL	NULL



Query window showing the following SQL code:

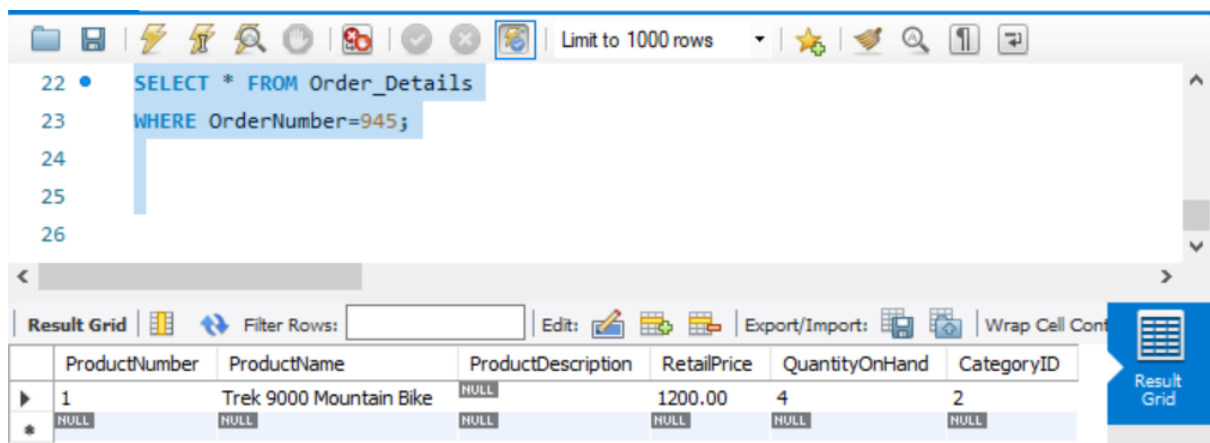
```
4 • SELECT * FROM Orders WHERE  
5   OrderNumber=945;  
6 • SELECT * FROM Order_Details  
7   WHERE OrderNumber=945;  
8
```

Result Grid:

OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
NULL	NULL	NULL	NULL

2) Still no change in output

3) Now all changes can be seen in their respective tables in right instance

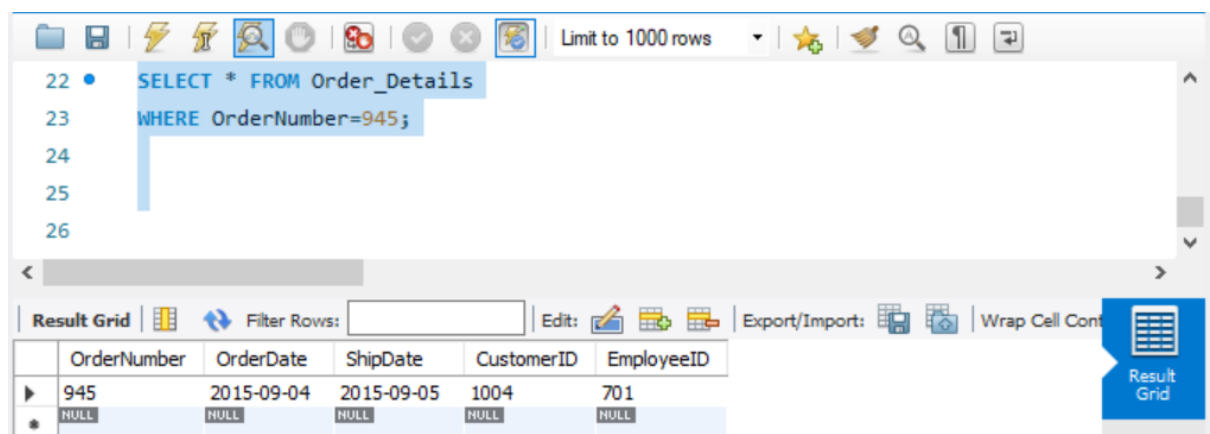


The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the following SQL query:

```
22 • SELECT * FROM Order_Details
23 WHERE OrderNumber=945;
24
25
26
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The grid has the following columns: ProductNumber, ProductName, ProductDescription, RetailPrice, QuantityOnHand, and CategoryID. The results are as follows:

	ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
▶	1	Trek 9000 Mountain Bike	NULL	1200.00	4	2
*	NULL	NULL	NULL	NULL	NULL	NULL

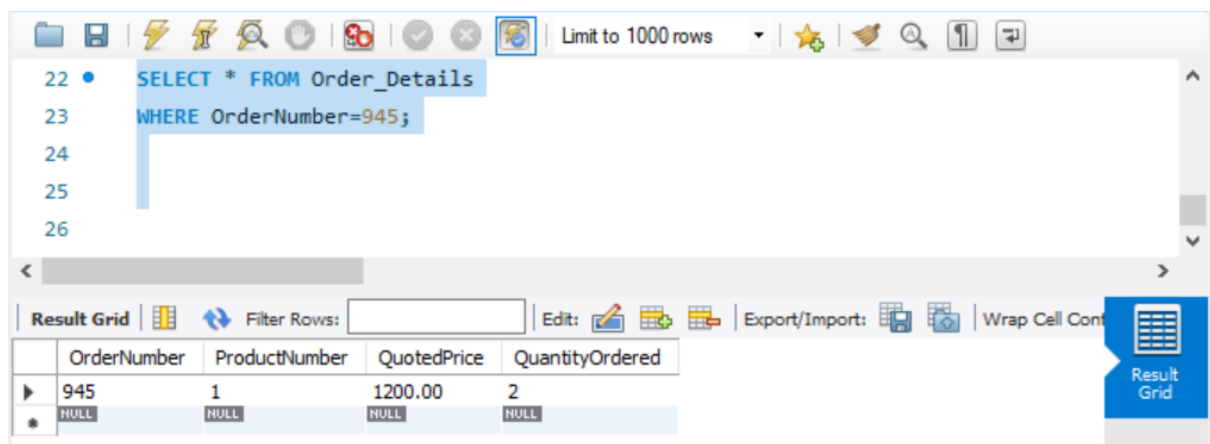


The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the following SQL query:

```
22 • SELECT * FROM Order_Details
23 WHERE OrderNumber=945;
24
25
26
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The grid has the following columns: OrderNumber, OrderDate, ShipDate, CustomerID, and EmployeeID. The results are as follows:

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



The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the following SQL query:

```
22 • SELECT * FROM Order_Details
23 WHERE OrderNumber=945;
24
25
26
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The grid has the following columns: OrderNumber, ProductNumber, QuotedPrice, and QuantityOrdered. The results are as follows:

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

4) Now all of the query results come as expected (giving the exact same results which had been seen in part 3)

Previously, the changes would only appear on transaction1 if it was committed in both transaction2 and transaction1. But now it can appear if only transaction 2 has been committed, but transaction1 has not!

This can lead to a lost update in the following way. At first the transaction1 read in the value before transaction2 wrote something to it. Thus it will not know the presence of the write part in transaction2. Thus when transaction1 will write something to it, it will have overridden the one written by transaction2's write. Thus when transaction1 commits, the transaction2 modification have been lost and hence a "lost update" has been formed.

Transaction1	Transaction2
R(k)	R(k)
W(k)	W(k)
C	C

Or in line syntax: R2(k) R1(k) W2(k) C2 W1(k)C1

Where r is read (ie any sort of select)

And w is write (ie any sort of update or insert)