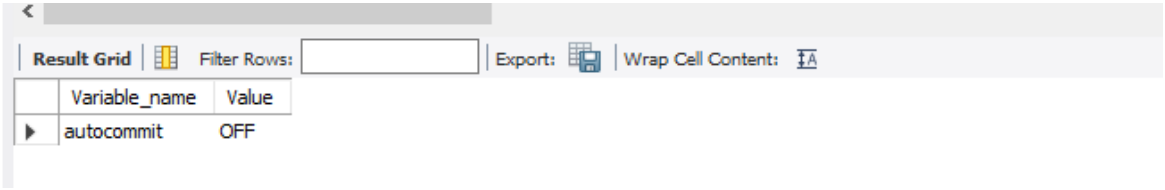
Fundamentals of Data Management

2019HS2 | 101624964 | Jimmy Trac

Credit Task 9.2.1

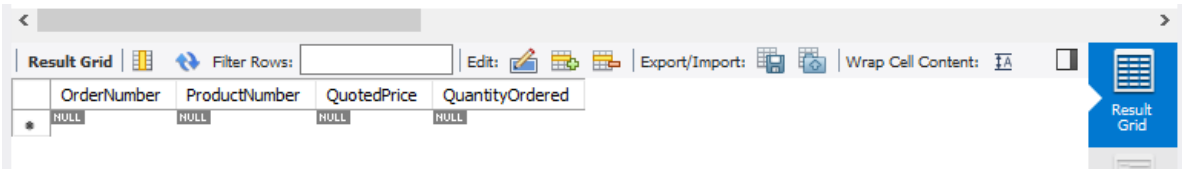
First making sure that `autocommit` is off:



a)

> *Run the first statement of T2 in your right MySQL Workbench instance. Run all statements of T1 in your left Workbench instance. What do you see?*

Result:

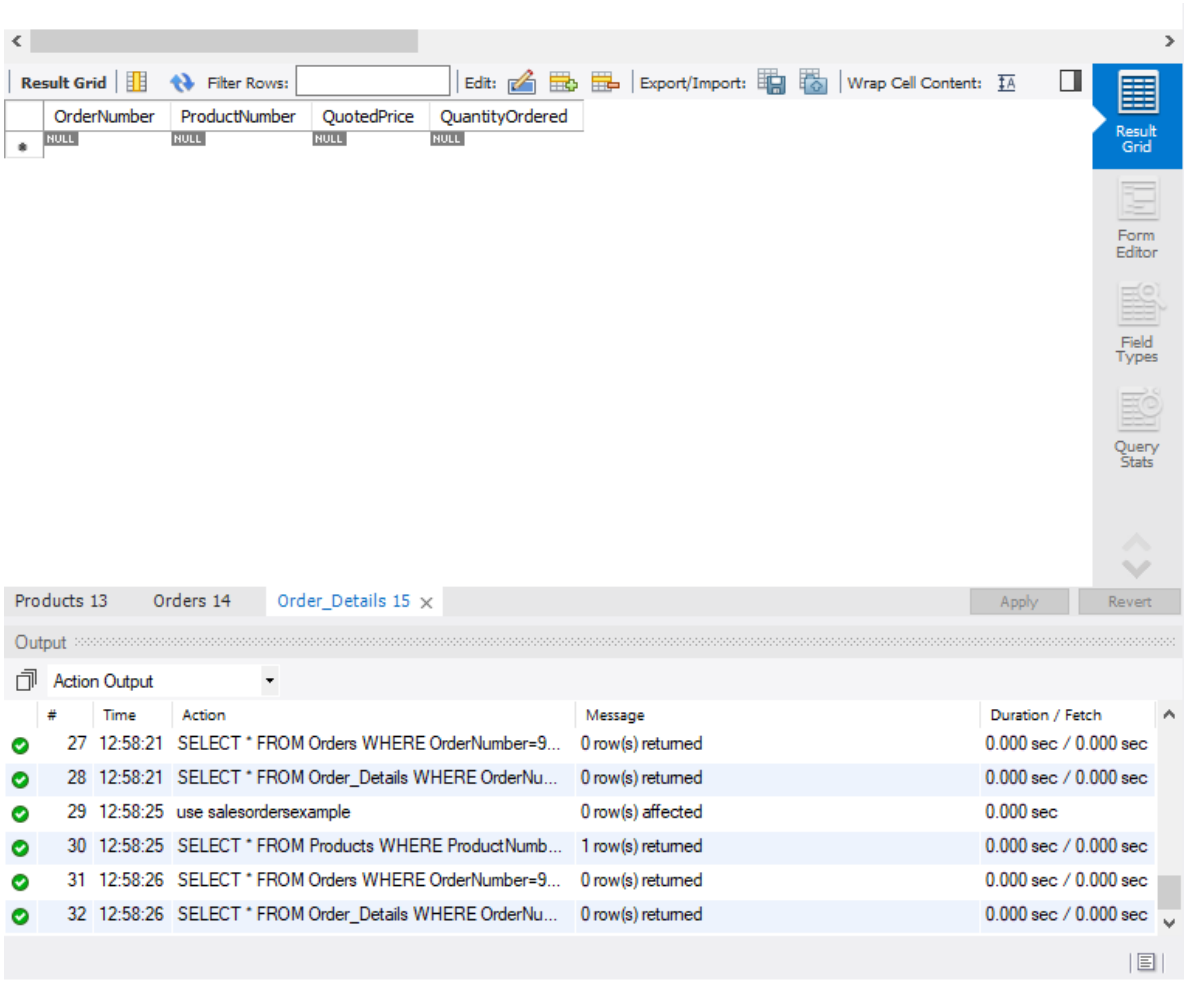


As the right MySQL workbench hasn't committed any of changes, we do not see any in the left instance.

b)

> *Run the rest of T2 in the right MySQL Workbench. Check again what you can see in your left Workbench.*

Result:



The above screenshot also shows the second use of the entire T1 script. Again, as there are no commits from T2, T1 returns null.

c)

> *Copy all statements of T1 into your right MySQL Workbench and run them. What do you see?*

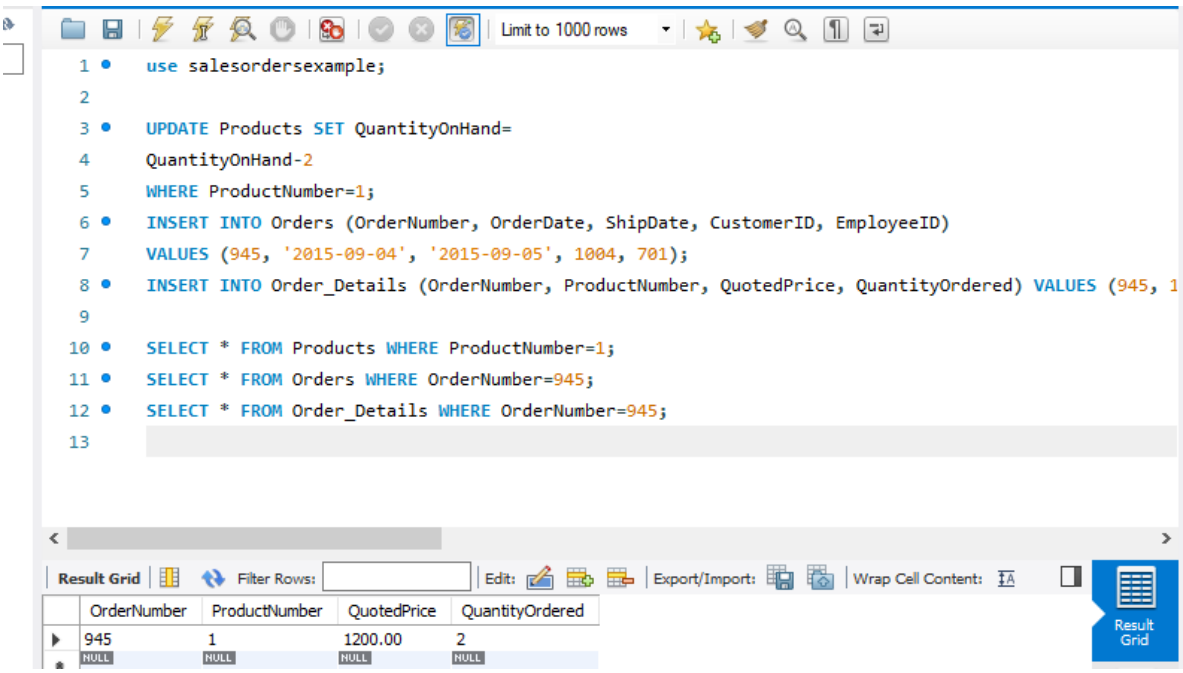
Running the following code:

```
use salesordersexample;

UPDATE Products SET QuantityOnHand=
QuantityOnHand-2
WHERE ProductNumber=1;
INSERT INTO Orders (OrderNumber, OrderDate, ShipDate, CustomerID, EmployeeID)
VALUES (945, '2015-09-04', '2015-09-05', 1004, 701);
INSERT INTO Order_Details (OrderNumber, ProductNumber, QuotedPrice, QuantityOrdered) VALUES (945, 1,
1200.00, 2);

SELECT * FROM Products WHERE ProductNumber=1;
SELECT * FROM Orders WHERE OrderNumber=945;
SELECT * FROM Order_Details WHERE OrderNumber=945;
```

Yields the following result:

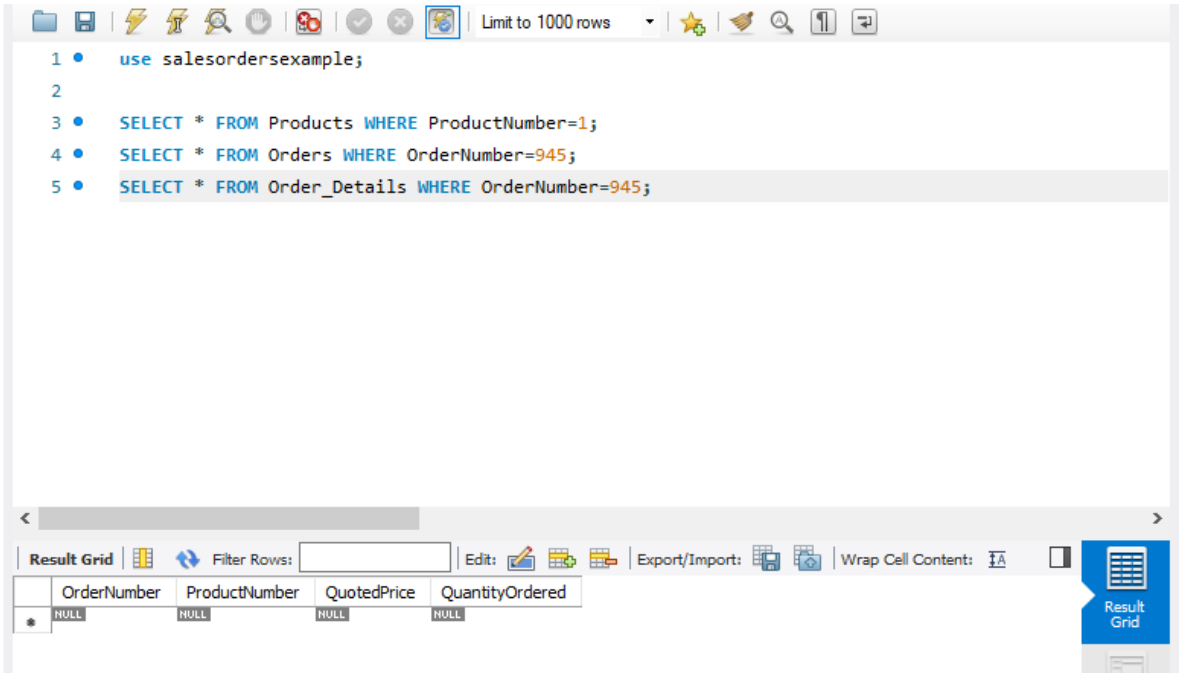


Showing that the T1 transaction is valid and functional in the T2 instance.

d)

> *Commit T2 in your right Workbench. Re-run T1 in your left instance. What do you see?*

Result:

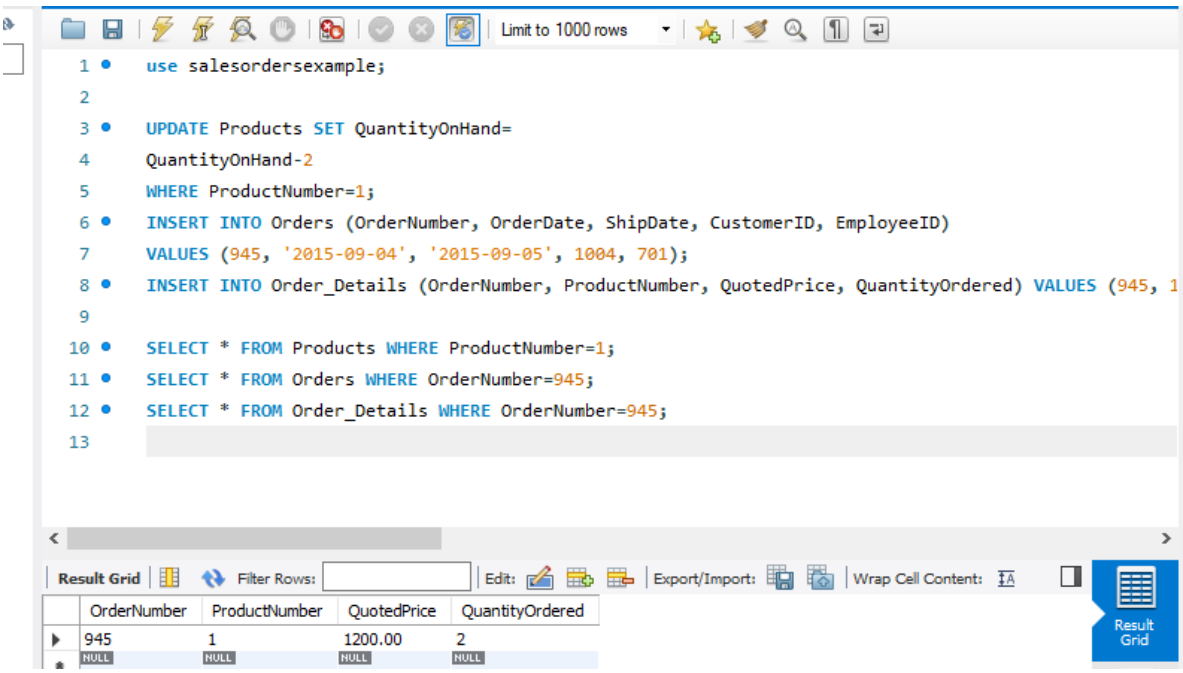


Again, `null` values.

e)

> *Commit T1 in your left Workbench. Re-run T1 again. What do you see?*

Result:



Questions

What isolation level are you working at? (You can check using: `show variables like 'tx_isolation';`)

Though the workbench won't provide a good answer (not returning `tx_isolation`), MySQL usually runs `REPEATABLE-READ`.

When does a transaction see the changes made? Answer for both the transaction that makes the changes and a transaction that merely reads the changes.

A transaction in `Repeatable-Read` can only see changes after the they have been committed.

In the case of a transaction that writes, it's written values can only be read **after commits**.

In the case of a transaction that reads, it can only read values that have been committed, and will not see any changes unless it commits itself.

Why can't T1 see the changes of T2 when T2 commits?

Within `Repeatable-Read`, transactions are unable to see changes once they start. This applies for **both** changes and reads. For example, Section D shows that T1 cannot read T2 unless T1 itself commits (Section E)

T1 cannot see T2 commits before T1 commits.

What do we mean by 'repeatable read' and do we have phantoms here in MySQL?

Repeatable-Read refers to the isolation level in MySQL. This is a matter of Concurrency, whereby the database product implements a level of Scheduling Policy to balance performance and isolation level (which maintains database integrity).

Phantoms are rows that appear during the middle of a *read* transaction due to another write transaction.

What does the SQL standard say about phantoms and Repeatable Read Isolation level?

The SQL `Repeatable-Read` isolation level does not allow value modifications during a transaction, however, does not prevent new rows from being created during a concurrent transaction. This means that the `Repeatable-Read` isolation level does allow for phantoms to appear whereas phantoms can only be eliminated at the `Serializable` isolation level.