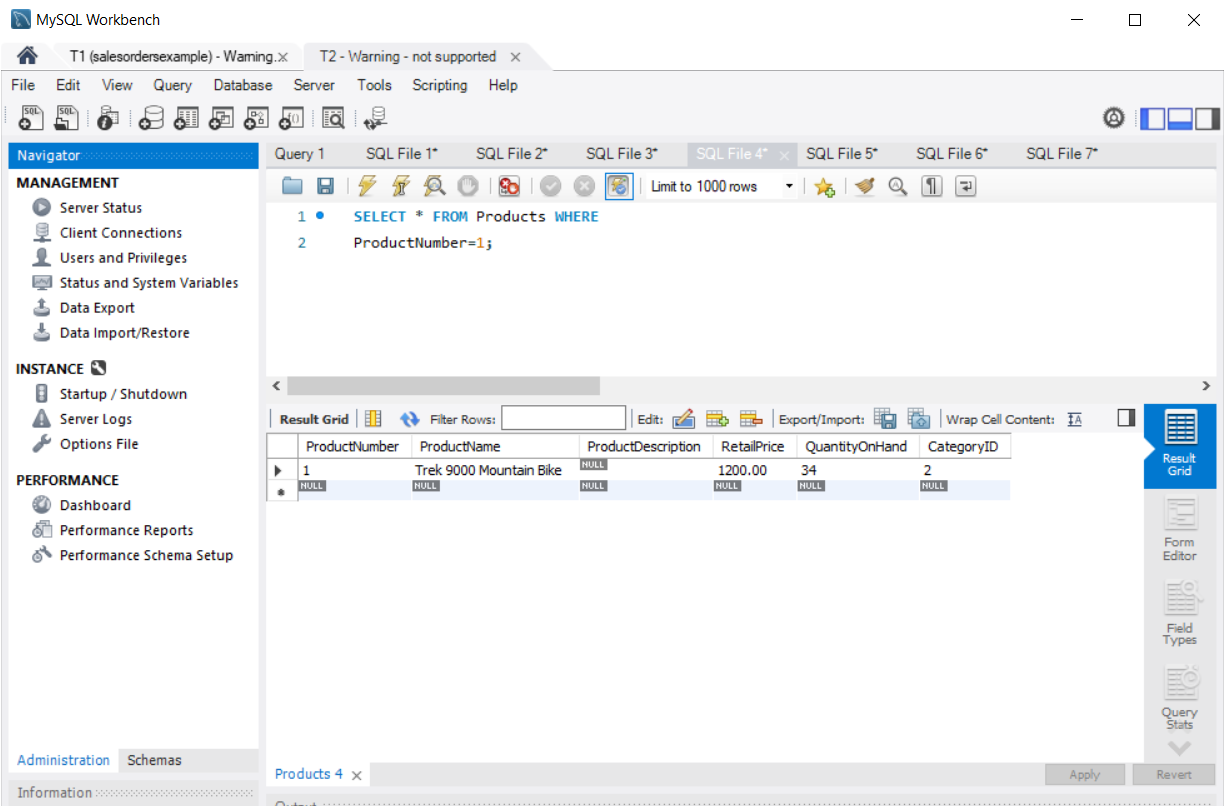
# Task C9.2.2

# Scenario 1:

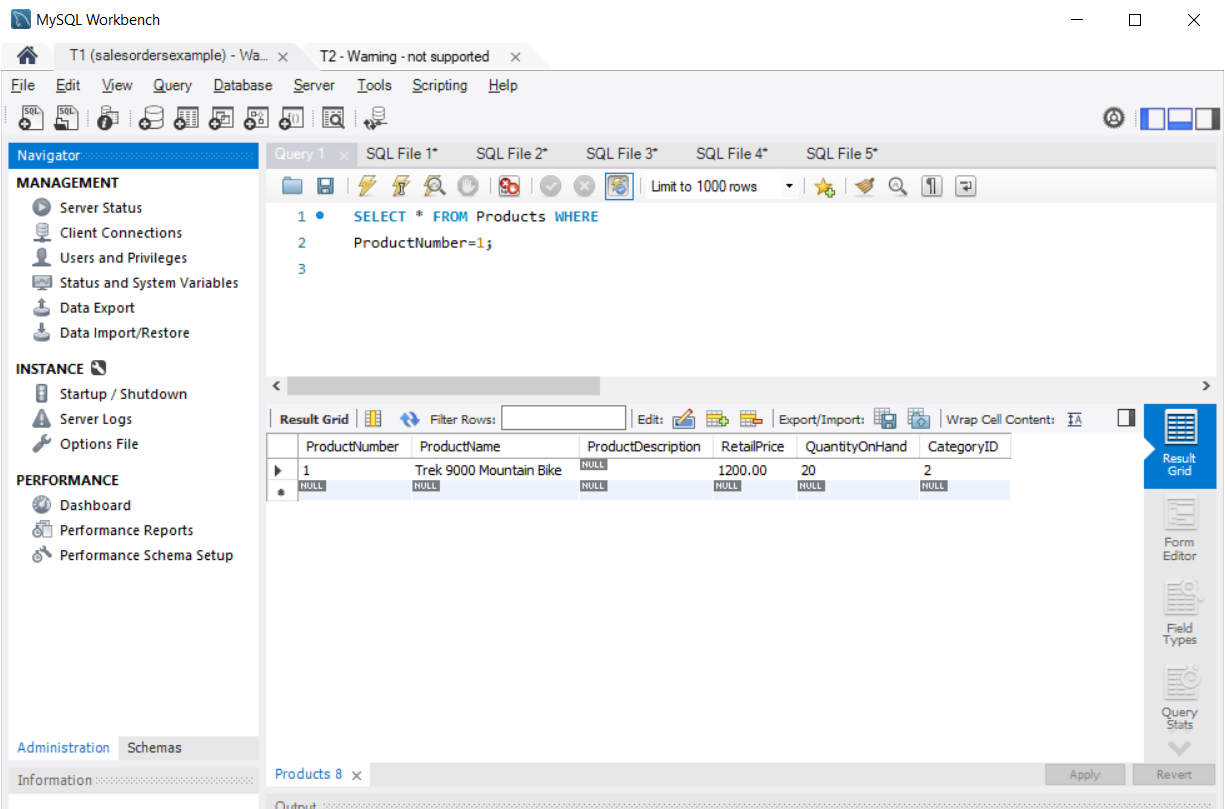
After executing statement “set session transaction isolation level read committed;” in both transactions.

If we execute first statement of T2 it immediately **shows changes in T2 but not in T1** as we have not committed.

Output in T2:



Output in T1:

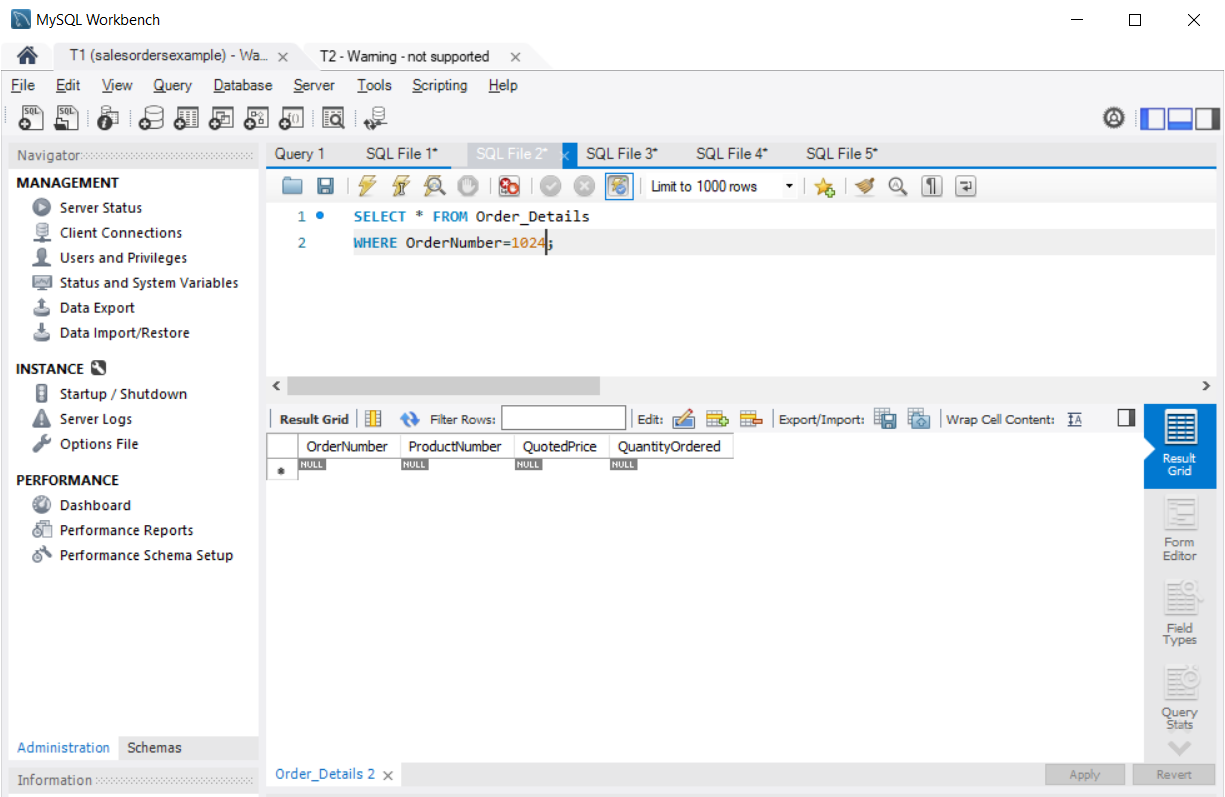


All other results for T1 are similar showing null results.

# Scenario 2:

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

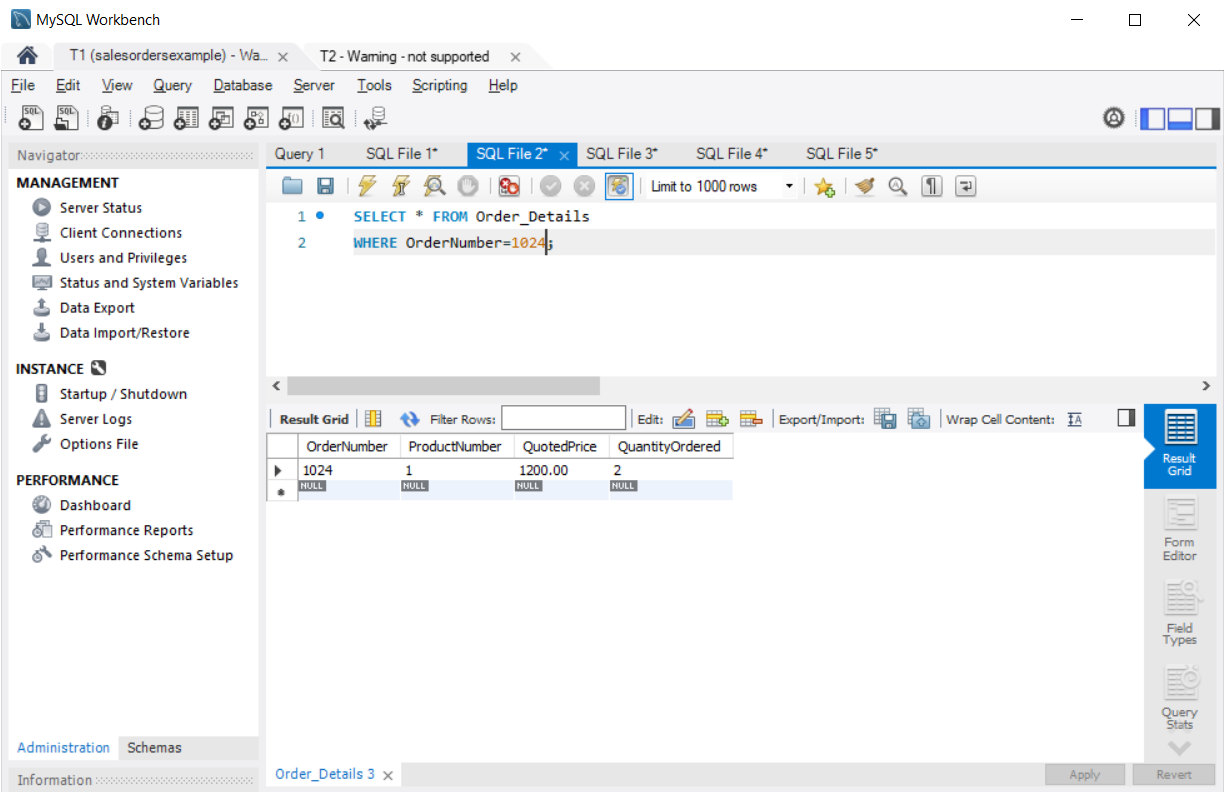
A: output in left Workbench is **still null** for provided insert statements in T2



# Scenario 3:

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

A: as soon as we run COMMIT in T2 changes get committed to database and become permanent. Those **changes starts to show immediately in T1** as well because it is in Read-Committed isolation level.



# Scenario 4:

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

A: Results are similar to previous scenario as we ware already able to see changes when COMMIT statement was executed in T2.

**Q: How did the query results differ from the ones in subtask 9.2.1?**

A: in previous task(with repeated read isolation), to reflect result in T1 we need to execute COMMIT command in both T1 and T2 but in this task(with read committed isolation) results got transferred to T1 even when only T2 executed COMMIT command. Just because of how isolation levels work.

**Q: How can this difference lead to a lost update? Explain the difference in your report and list the necessary SQL statements to produce a lost update at read committed isolation level.**

A: because of how Read Committed isolation works by just one side committing changes can lead to potentially harmful consequences, one of which is lost update. It accures when two or more transections are updating same set of rows simultaneously and committing at the same time.

In this case transection committing at last overwrites previous updates to that row(s) and results in list updates for other transections.

For example in given student table;

Transection 1

|  |  |
| --- | --- |
| Transection 1 | Transection 2 |
| UPDATE Students SET grades = 30 WHERE studentid = 123;  COMMIT; | UPDATE Students SET grades = 40 WHERE studentid = 123; |
|  | COMMIT: |
| SELECT \* FROM Students WHERE studentid = 123; | SELECT \* FROM Students WHERE studentid = 123; |

In this case result of both transections will show grades = 40 because Transection 2 is executing COMMIT statement after Transection 1. it overwrites changes made to shared row(s) by Transection 1.