

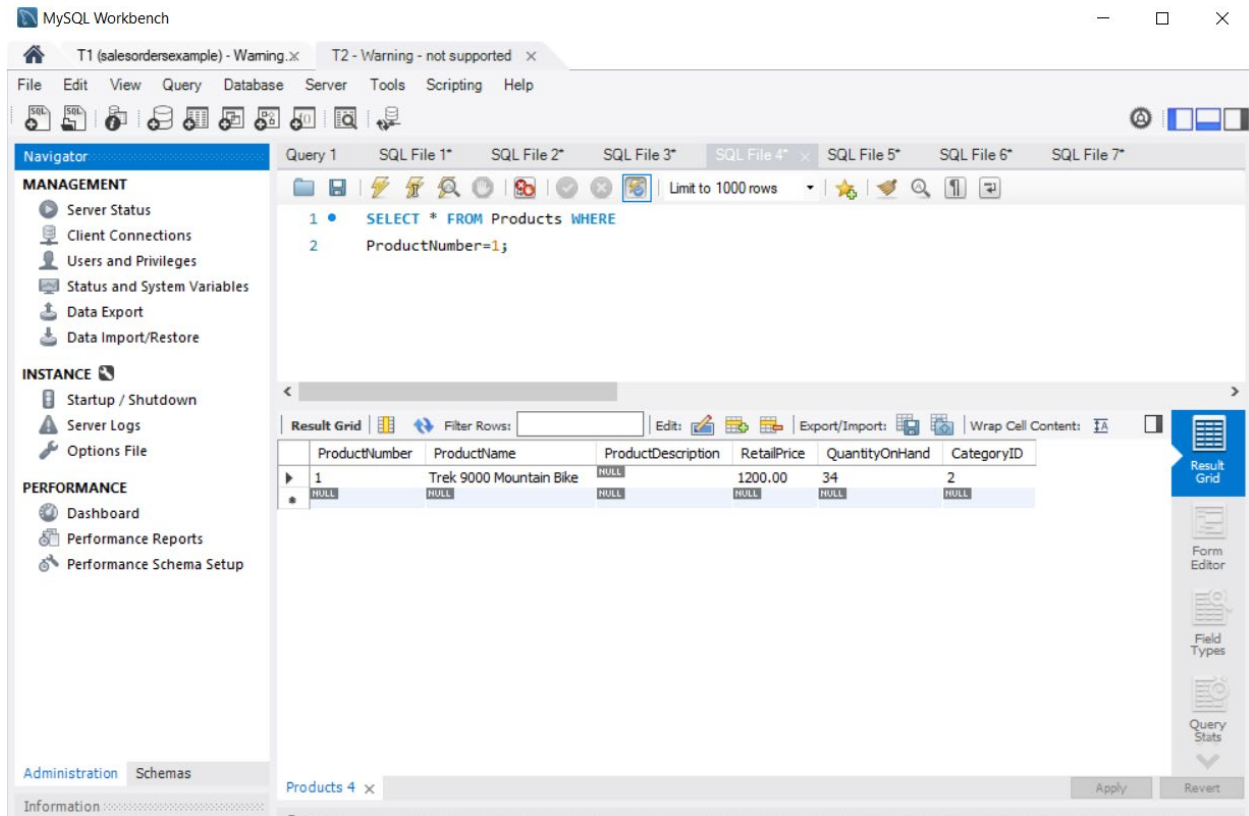
Task C9.2.2

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:



The screenshot shows the MySQL Workbench interface. The left sidebar contains a 'Navigator' panel with sections for 'MANAGEMENT' (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), 'INSTANCE' (Startup / Shutdown, Server Logs, Options File), and 'PERFORMANCE' (Dashboard, Performance Reports, Performance Schema Setup). The bottom status bar shows 'Administration' and 'Schemas' tabs, with 'Products 4' selected. The main query editor displays a SQL query: `SELECT * FROM Products WHERE ProductNumber=1;`. The 'Result Grid' at the bottom shows the query results in a table format.

ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
1	Trek 9000 Mountain Bike	NULL	1200.00	34	2
NULL	NULL	NULL	NULL	NULL	NULL

Output in T1:

The screenshot shows the MySQL Workbench interface. The left sidebar contains a 'Navigator' panel with sections for 'MANAGEMENT' (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), 'INSTANCE' (Startup / Shutdown, Server Logs, Options File), and 'PERFORMANCE' (Dashboard, Performance Reports, Performance Schema Setup). The bottom of the sidebar has tabs for 'Administration' and 'Schemas'. The main window displays a query in the 'Query Editor' with the following SQL:

```
1 • SELECT * FROM Products WHERE
2   ProductNumber=1;
3
```

The 'Result Grid' shows the output of the query. It has columns: ProductNumber, ProductName, ProductDescription, RetailPrice, QuantityOnHand, and CategoryID. The first row shows the product details for ProductNumber 1. The second row, marked with an asterisk, shows null values for all columns.

ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
1	Trek 9000 Mountain Bike	NULL	1200.00	20	2
*	NULL	NULL	NULL	NULL	NULL

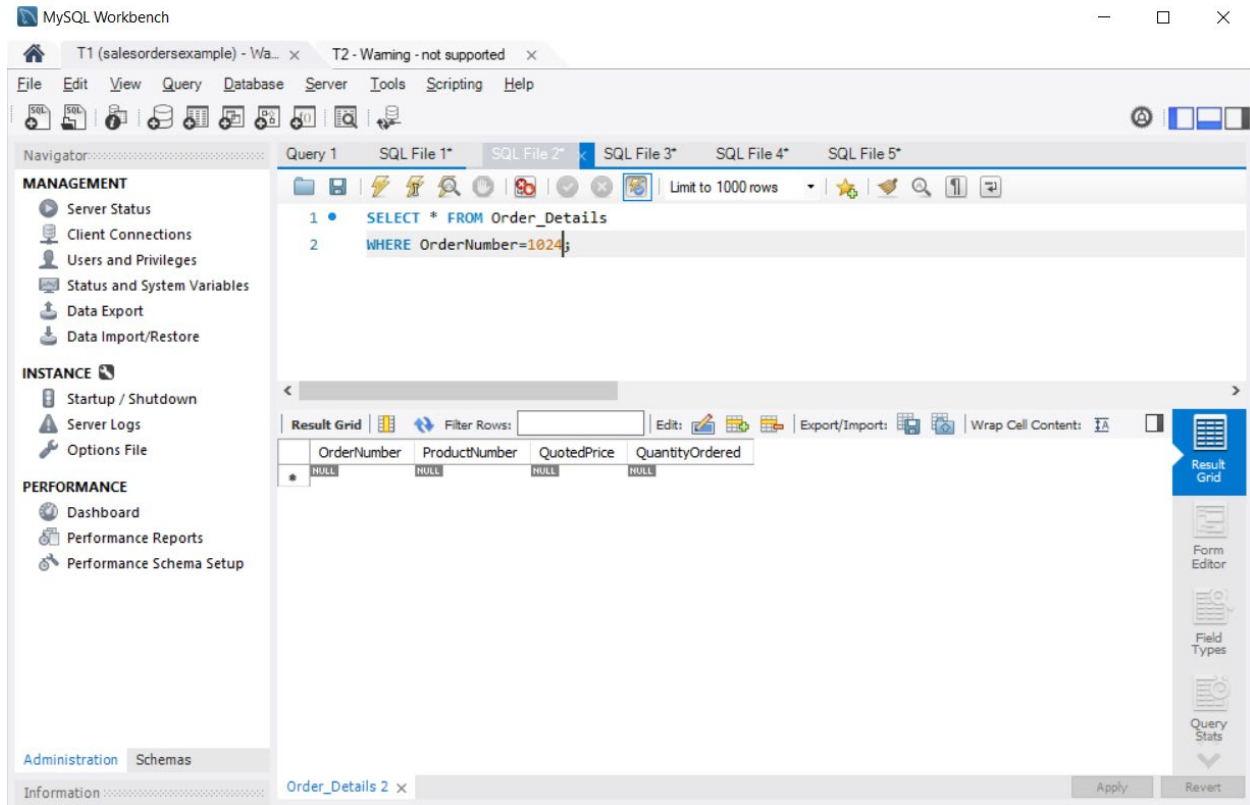
The right sidebar contains a 'Result Grid' button and a 'Query Stats' button. The bottom status bar shows 'Products 8' and 'Apply' and 'Revert' buttons.

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.

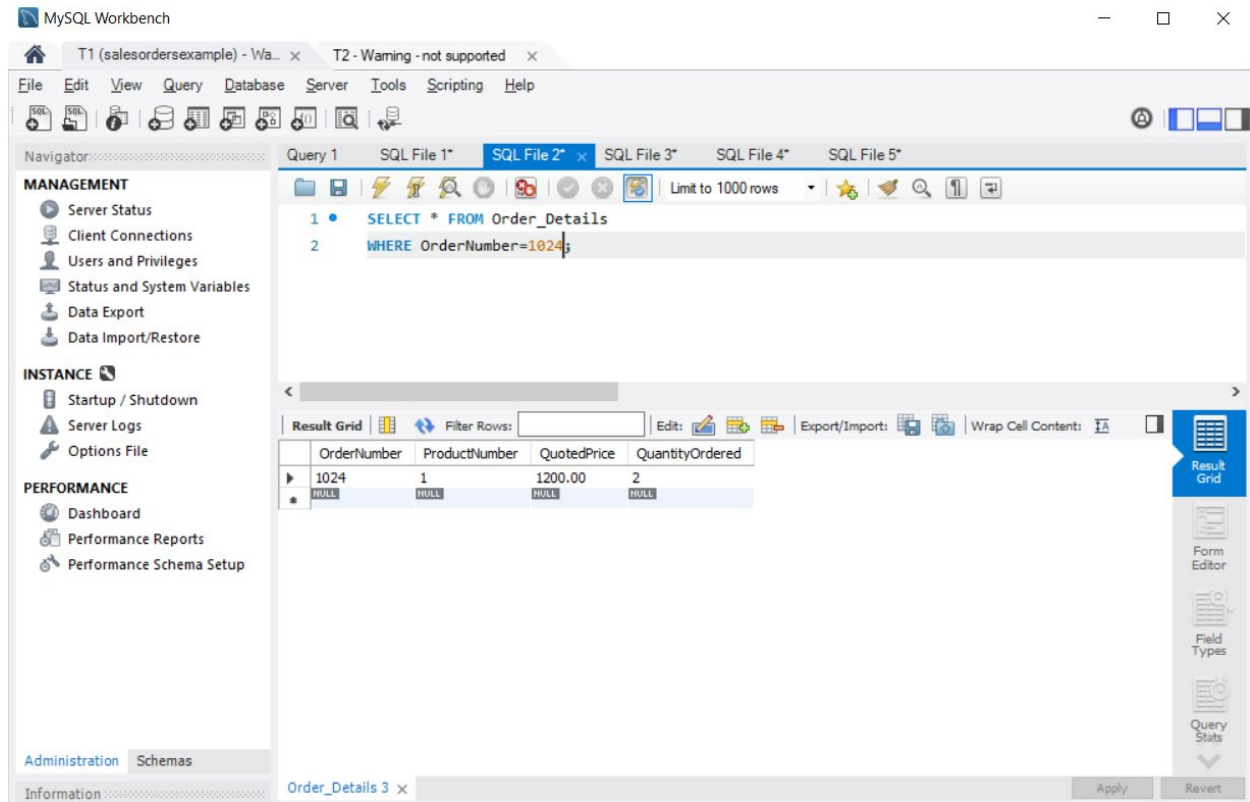
A: output in left Workbench is **still null** for provided insert statements in T2



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 were 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 occurs when two or more transactions are updating same set of rows simultaneously and committing at the same time.

In this case transaction committing at last overwrites previous updates to that row(s) and results in lost updates for other transactions.

For example in given student table;

Transaction 1

Transaction 1	Transaction 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 transactions will show grades = 40 because Transaction 2 is executing COMMIT statement after Transaction 1. It overwrites changes made to shared row(s) by Transaction 1.