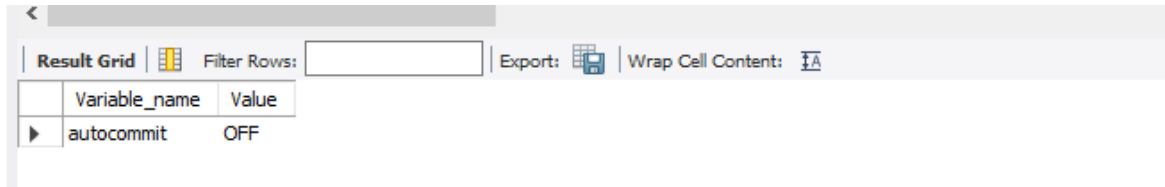


Fundamentals of Data Management

2019HS2 | 101624964 | Jimmy Trac

Credit Task 9.2.1

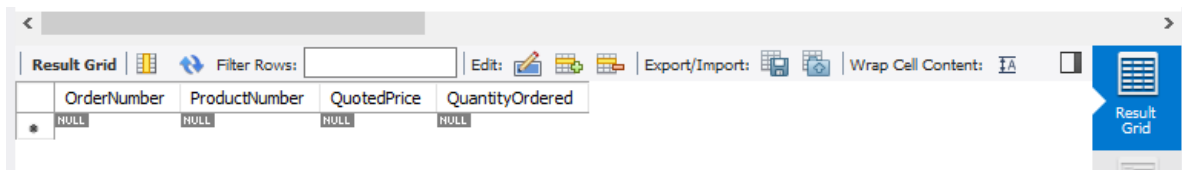
First making sure that `autocommit` is off:



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 screenshot shows the MySQL Workbench interface. At the top, the 'Result Grid' tab is active, displaying a table with four columns: OrderNumber, ProductNumber, QuotedPrice, and QuantityOrdered. All four columns contain the value 'NULL'. Below this, the 'Output' tab is active, showing the 'Action Output' log. The log contains six entries, each with a green checkmark icon, indicating successful execution. The entries are as follows:

#	Time	Action	Message	Duration / Fetch
27	12:58:21	SELECT * FROM Orders WHERE OrderNumber=9...	0 row(s) returned	0.000 sec / 0.000 sec
28	12:58:21	SELECT * FROM Order_Details WHERE OrderNu...	0 row(s) returned	0.000 sec / 0.000 sec
29	12:58:25	use salesordersexample	0 row(s) affected	0.000 sec
30	12:58:25	SELECT * FROM Products WHERE ProductNumb...	1 row(s) returned	0.000 sec / 0.000 sec
31	12:58:26	SELECT * FROM Orders WHERE OrderNumber=9...	0 row(s) returned	0.000 sec / 0.000 sec
32	12:58:26	SELECT * FROM Order_Details WHERE OrderNu...	0 row(s) returned	0.000 sec / 0.000 sec

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:

```

1  use salesordersexample;
2
3  UPDATE Products SET QuantityOnHand=
4  QuantityOnHand-2
5  WHERE ProductNumber=1;
6  INSERT INTO Orders (OrderNumber, OrderDate, ShipDate, CustomerID, EmployeeID)
7  VALUES (945, '2015-09-04', '2015-09-05', 1004, 701);
8  INSERT INTO Order_Details (OrderNumber, ProductNumber, QuotedPrice, QuantityOrdered) VALUES (945, 1,
9  1200.00, 2);
10
11 SELECT * FROM Products WHERE ProductNumber=1;
12 SELECT * FROM Orders WHERE OrderNumber=945;
13 SELECT * FROM Order_Details WHERE OrderNumber=945;

```

Yields the following result:

The screenshot shows a SQL query window with the following code:

```

1 • use salesordersexample;
2
3 • UPDATE Products SET QuantityOnHand=
4   QuantityOnHand-2
5   WHERE ProductNumber=1;
6 • INSERT INTO Orders (OrderNumber, OrderDate, ShipDate, CustomerID, EmployeeID)
7   VALUES (945, '2015-09-04', '2015-09-05', 1004, 701);
8 • INSERT INTO Order_Details (OrderNumber, ProductNumber, QuotedPrice, QuantityOrdered) VALUES (945, 1
9
10 • SELECT * FROM Products WHERE ProductNumber=1;
11 • SELECT * FROM Orders WHERE OrderNumber=945;
12 • SELECT * FROM Order_Details WHERE OrderNumber=945;
13

```

Below the query window is a result grid with the following data:

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

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:

The screenshot shows a SQL query window with the following code:

```

1 • use salesordersexample;
2
3 • SELECT * FROM Products WHERE ProductNumber=1;
4 • SELECT * FROM Orders WHERE OrderNumber=945;
5 • SELECT * FROM Order_Details WHERE OrderNumber=945;

```

Below the query window is a result grid with the following data:

	OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
*	NULL	NULL	NULL	NULL

Again, null values.

e)

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

Result:

The screenshot shows a MySQL Workbench window with a SQL editor and a result grid. The SQL editor contains the following queries:

```

1 • use salesordersexample;
2
3 • UPDATE Products SET QuantityOnHand=
4   QuantityOnHand-2
5   WHERE ProductNumber=1;
6 • INSERT INTO Orders (OrderNumber, OrderDate, ShipDate, CustomerID, EmployeeID)
7   VALUES (945, '2015-09-04', '2015-09-05', 1004, 701);
8 • INSERT INTO Order_Details (OrderNumber, ProductNumber, QuotedPrice, QuantityOrdered) VALUES (945, 1
9
10 • SELECT * FROM Products WHERE ProductNumber=1;
11 • SELECT * FROM Orders WHERE OrderNumber=945;
12 • SELECT * FROM Order_Details WHERE OrderNumber=945;
13

```

The result grid shows the following data:

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

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.