|  |  |  |  |
| --- | --- | --- | --- |
| Tên thủ tục | Mục đích | Input | Output |
| sp\_CapNhatSoPhanAn | Để cập nhật số lượng của một món tại một chi nhánh trong một ngày nhất định | @MACN, @MAMON, @NGAY ,@SOLUONG, @SL\_CON | Cập nhật dữ liệu vào bảng thực đơn |
| sp\_ThongKeTheoNgayCN | Thống kê doanh thu theo từng ngày của một chi nhánh nhất định | @MACN | Gồm ngày và doanh thu trong ngày đó |
| sp\_ThongKeTheoThangCN | Thống kê doanh thu theo từng tháng của một chi nhánh nhất định | @MACN | Gồm tháng, năm và doanh thu trong tháng đó |
| sp\_ThongKeTheoNamCN | Thống kê doanh thu theo từng năm của một chi nhánh nhất định | @MACN | Gồm năm và doanh thu trong năm đó |
| sp\_ThongKeTheoMonCN | Thống kê doanh thu theo món của một chi nhánh nhất định | @MACN | Gồm món và doanh thu của món đó |
| sp\_ThongKeTheoLoaiMonCN | Thống kê doanh thu theo loại món của một chi nhánh nhất định | @MACN | Gồm loại món và doanh thu của loại món đó |
| sp\_TheoKenhDatHangCN | Thống kê doanh thu theo kênh đặt hàng của một chi nhánh nhất định | @MACN | Gồm kênh đặt hàng và doanh thu của kênh đó |
| sp\_ThongKeTheoLoaiKhachHangCN | Thống kê doanh thu theo loại khách hàng của một chi nhánh nhất định | @MACN | Gồm 2 cột là khách vãng lai và thành viên chứa doanh thu của từng loại |
| sp\_ThongKeTheoThangTong | Thống kê doanh thu theo từng tháng của toàn chi nhánh |  | Gồm tháng, năm và doanh thu trong tháng đó |
| sp\_ThongKeTheoNamTong | Thống kê doanh thu theo từng năm của toàn chi nhánh |  | Gồm năm và doanh thu trong năm đó |
| sp\_ThongKeTheoMonTong | Thống kê doanh thu theo từng món của toàn chi nhánh |  | Gồm tên món và doanh thu của món đó |
| sp\_ThongKeTheoLoaiMonTong | Thống kê doanh thu theo từng loại món của toàn chi nhánh |  | Gồm tên loại món và doanh thu của loại món đó |
| sp\_ThongKeTheoNgayTong | Thống kê doanh thu theo từng ngày của toàn chi nhánh |  | Gồm ngày và doanh thu trong ngày đó |
| sp\_TheoKenhDatHangTong | Thống kê doanh thu theo kênh đặt hàng của toàn chi nhánh |  | Gồm kênh đặt hàng và doanh thu của kênh đó |
| sp\_ThongKeTheoLoaiKhachHangTong | Thống kê doanh thu theo loại khách hàng của toàn chi nhánh |  | Gồm khách vãng lai và thành viên |

* ***Thủ tục tranh chấp:***

|  |  |  |
| --- | --- | --- |
| Tên lỗi | Kịch bản lỗi | Xử lý |
| Dirty Read | Gỉa sử nhân viên 1 thêm món vào thực đơn, nhân viên 2 xem thực đơn thì sẽ bị đọc lỗi nếu giao tác T1 không commit mà rollback lại. | Thiết lập mức cô lập Read Committed |
| Unrepeatable Read | T1 xem trong thực đơn, sau đó T2 xóa trong phần T1 đã xem sau đó T1 xem lại thì kết quả không giống nhau | Thiết lập mức cô lập Repeatable Read |
| Lost Update | T1 đăng nhập và thay đổi giá trị thẻ, sau đó T2 đăng nhập và cũng thay đổi giá trị thẻ. Qua đó update của T1 bị mất | Thiết lập mức cô lập Serializable |
| Phantom | Gỉa sử T1 thống kê vào ngày hôm nay tại chi nhánh 1. Sau đó T2 insert vào bảng đơn hàng. T1 thống kê lại thì thấy thêm dữ liệu T2 vừa thêm vào dù chưa commit. Lỗi phantom. | Thiết lập mức cô lập Serializable |
| Deadlock | Gỉa sử có hai người quản lý công ty, T1 thêm món vào thực đơn chi nhánh 1 và xem thực đơn chi nhánh 2, T2 thêm món vào thực đơn chi nhánh 2 và xem món chi nhánh 1. | SQL sẽ hủy 1 giao tác và làm việc tuần tự. |

* ***Chi tiết thủ tục tranh chấp:***

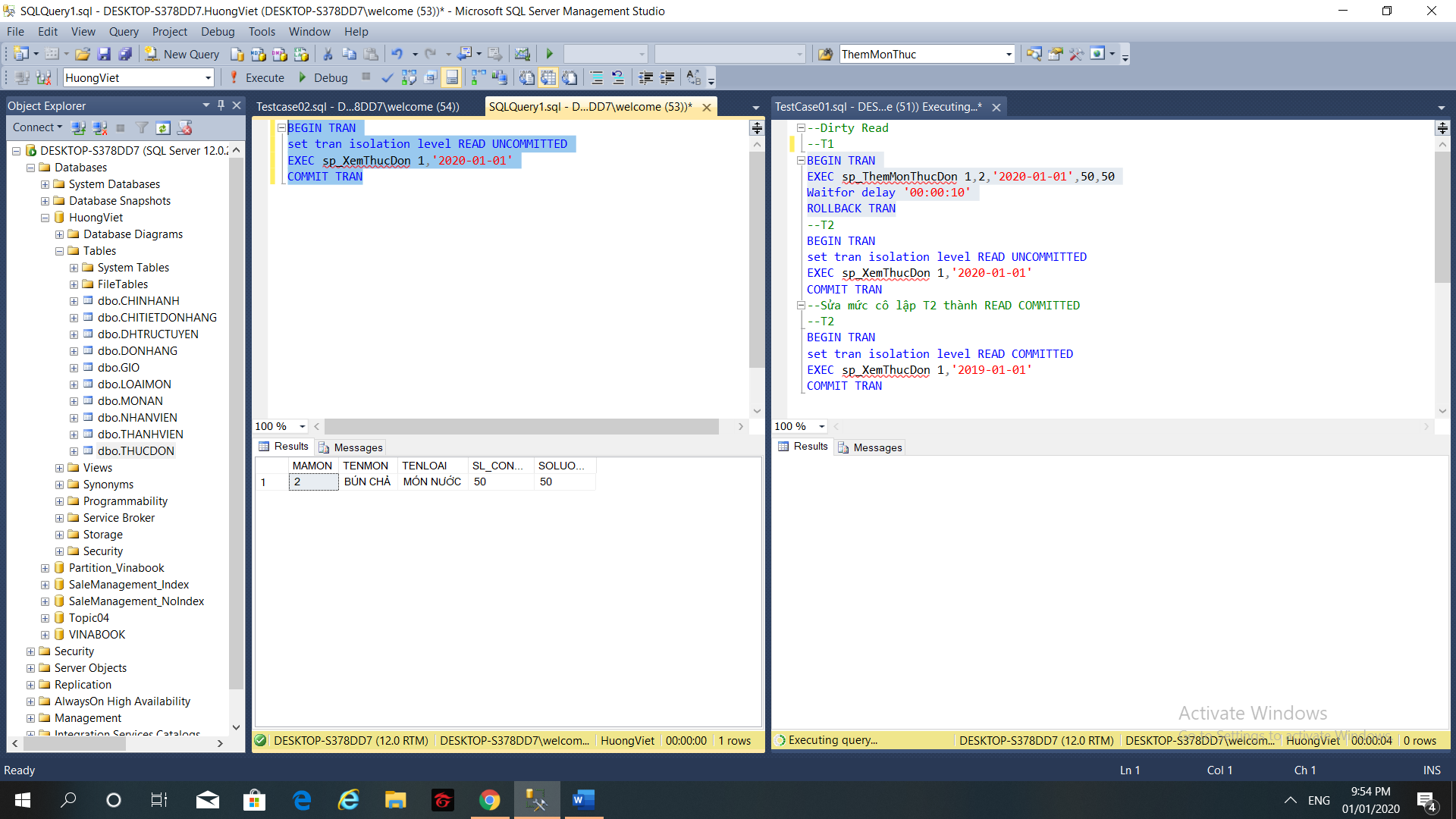
1. ***Dirty read:***
2. ***Lỗi :*** Gỉa sử nhân viên 1 thêm món vào thực đơn, nhân viên 2 xem thực đơn thì sẽ bị đọc lỗi nếu giao tác T1 không commit mà rollback lại. Ở ví dụ này, để xảy ra lỗi thì giả lập mức cô lập cho giao tác T2 là Read Uncommitted

|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_ThemMonThucDon 1,2,'2020-01-01',50,50  Waitfor delay '00:00:10'  ROLLBACK TRAN | BEGIN TRAN  set tran isolation level READ UNCOMMITTED  EXEC sp\_XemThucDon 1,'2020-01-01'  COMMIT TRAN |

1. Chạy lỗi trên SQL:

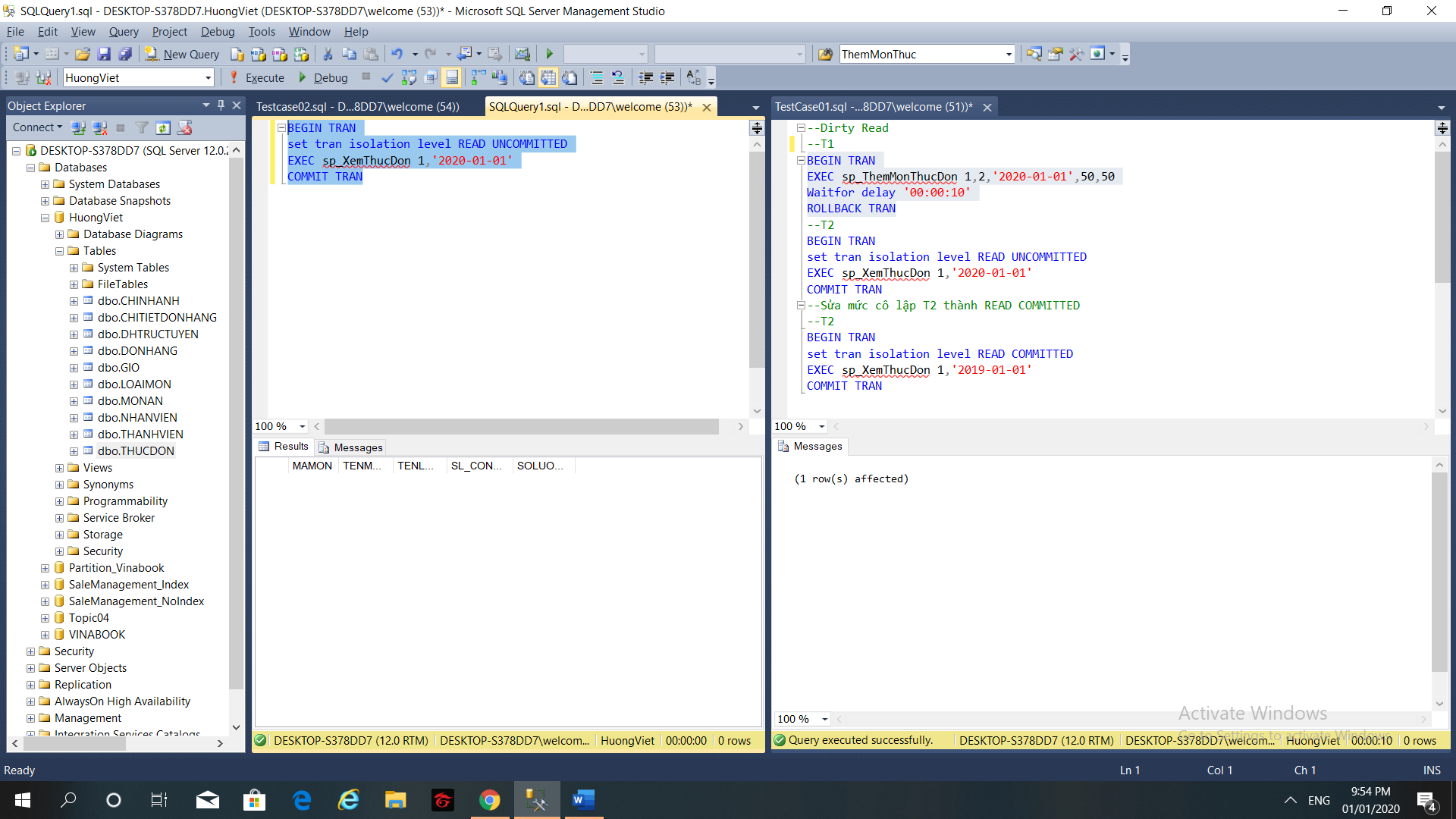
**Đoạn test case của trường hợp Dirty Read:**

Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:



* Giao tác 2 đã đọc dữ liệu mà giao tác 1 đã cập nhật nhưng chưa rollback với nội dung thêm 1 món vào thực đơn.

Sau khi Giao tác 1 rollback, thử truy vấn lại T2. Ta thấy kết quả đã không còn giống ban đầu.

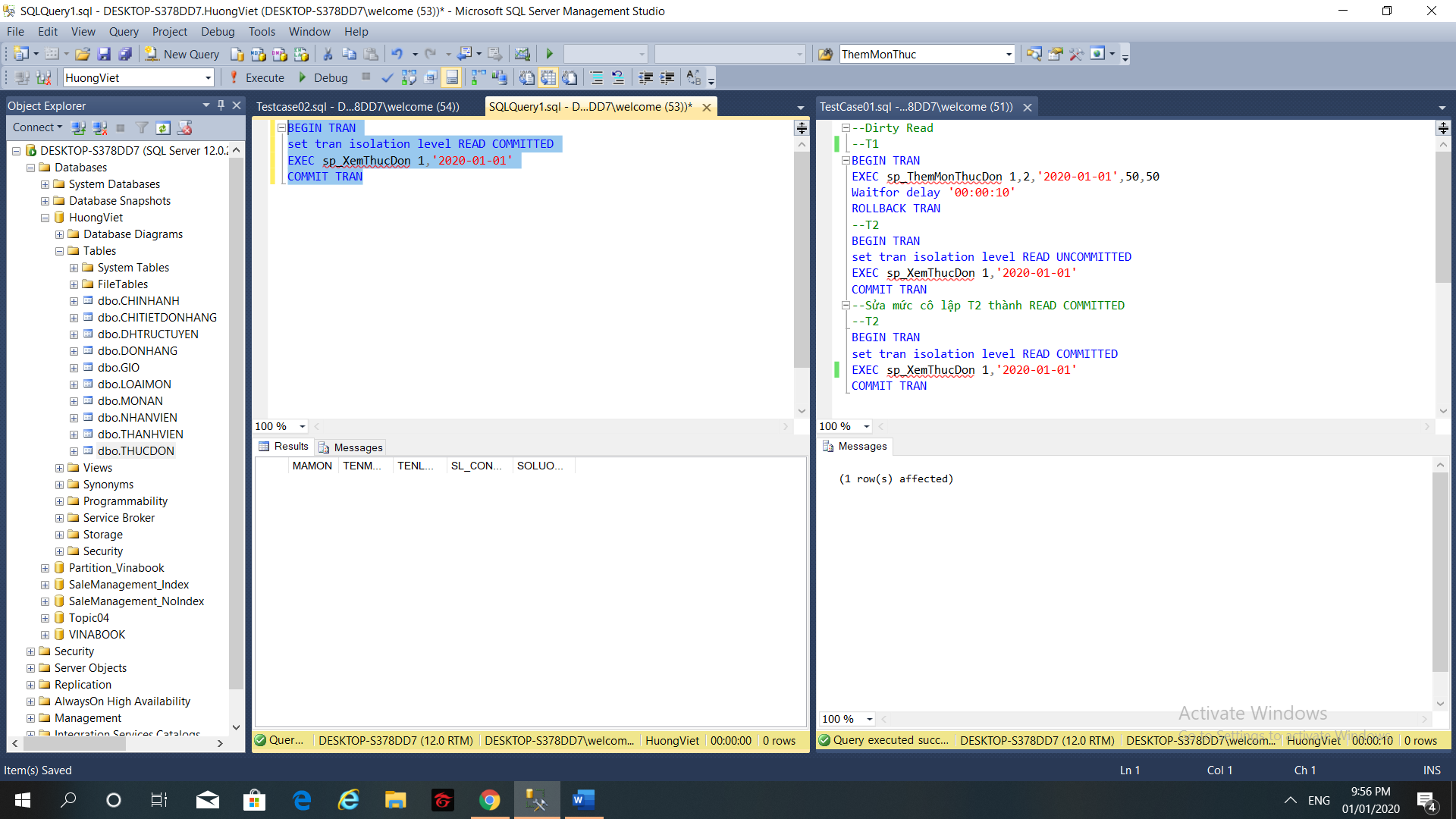


1. ***Giải pháp:*** Set cơ chế khóa Read Committed

Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2, lúc này T2 không đọc được dữ liệu mà T1 thay đổi do T1 vẫn chưa kết thúc giao tác.



Sau khi T1 rollback, T2 lập tức thu được dữ liệu ban đầu.

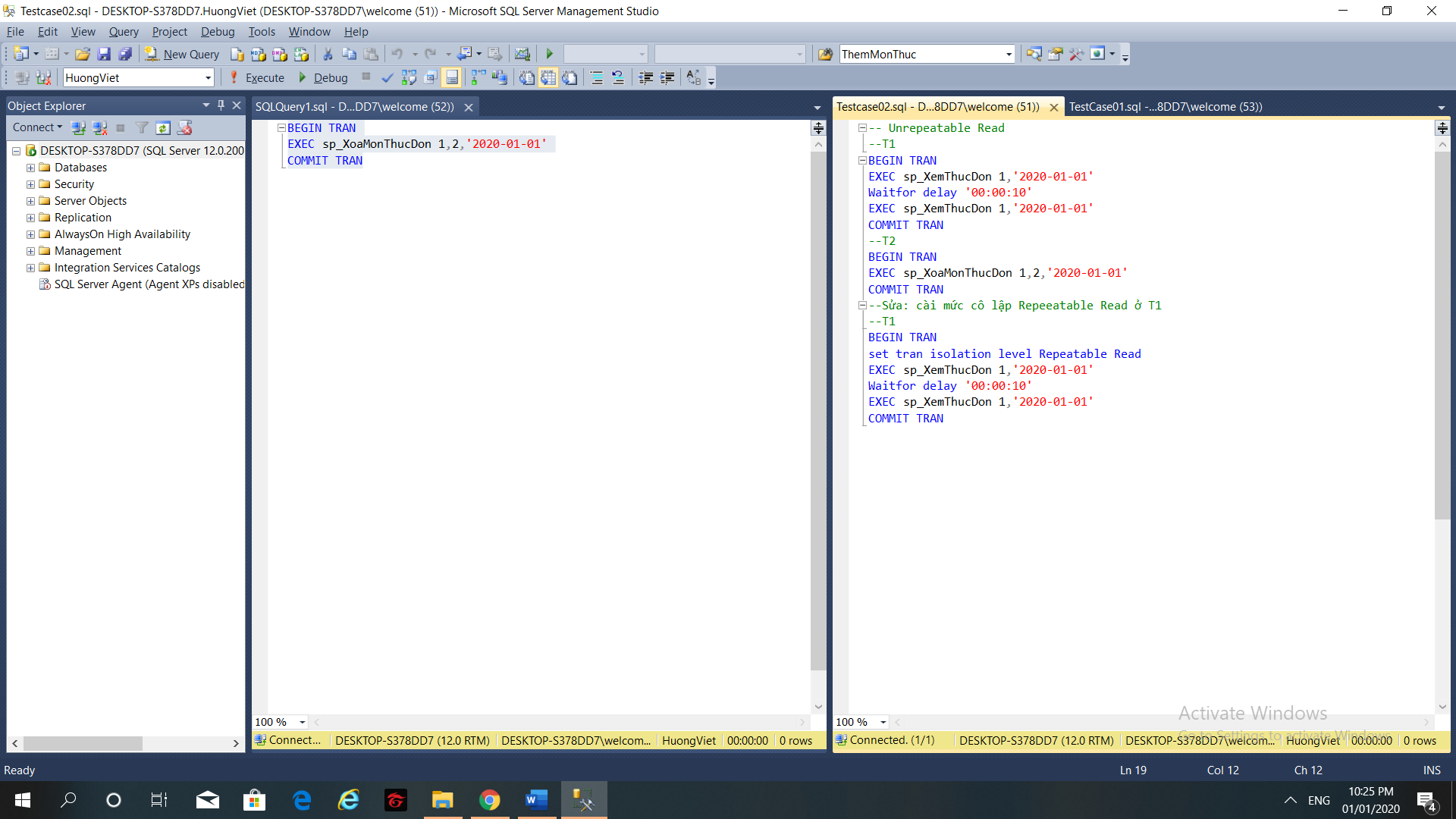


1. ***Unrepeatable read:***
2. ***Lỗi 1:*** T1 xem trong thực đơn, sau đó T2 xóa trong phần T1 đã xem sau đó T1 xem lại thì kết quả không giống nhau

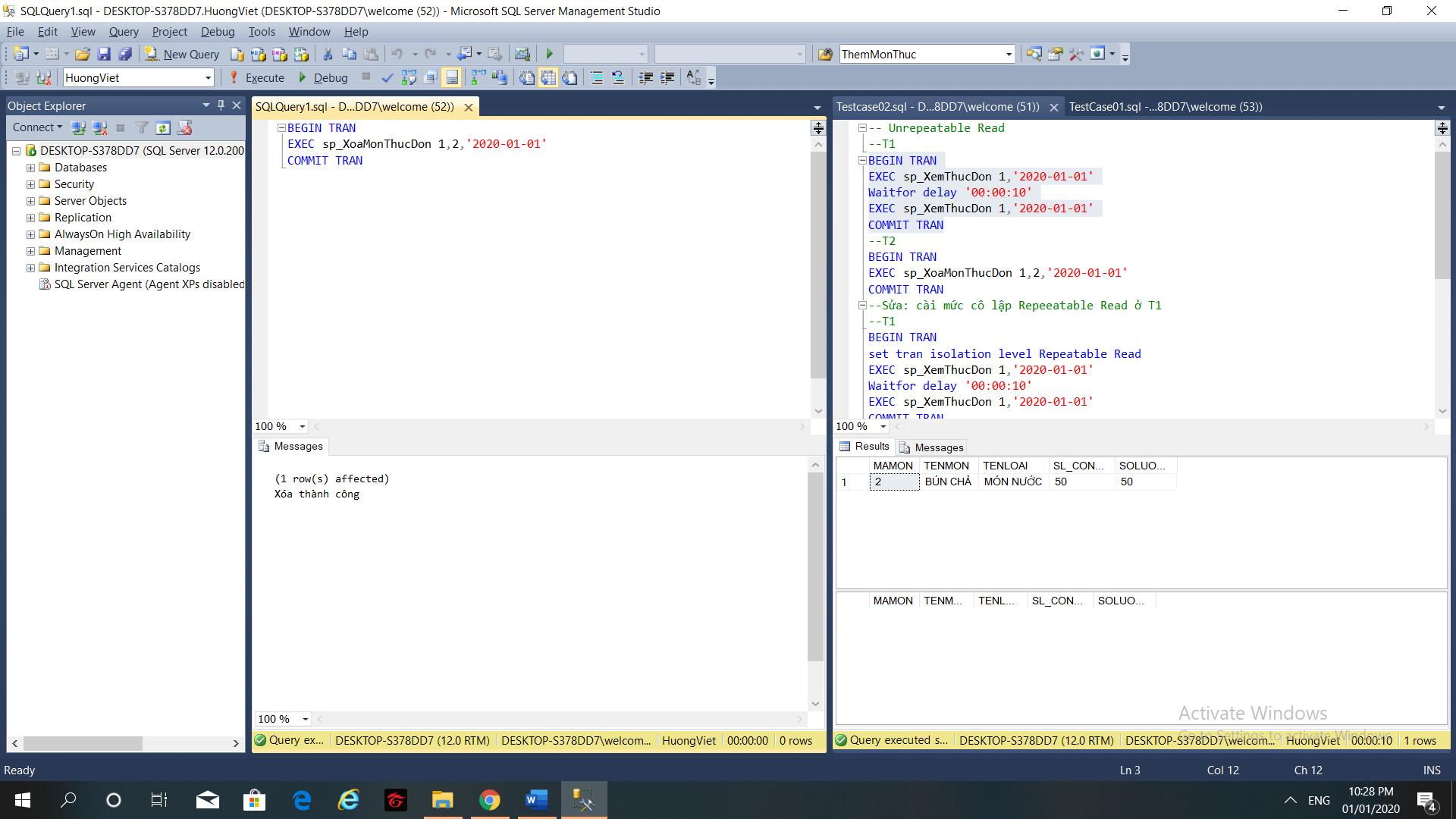
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_XemThucDon 1,'2020-01-01'  Waitfor delay '00:00:10'  EXEC sp\_XemThucDon 1,'2020-01-01'  COMMIT TRAN | BEGIN TRAN  EXEC sp\_XoaMonThucDon 1,2,'2020-01-01'  COMMIT TRAN |

1. Chạy lỗi trên SQL

**Đoạn test case của trường hợp Unrepeatable Read: (Giả lập lấy cơ chế Read Committed)**



Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:



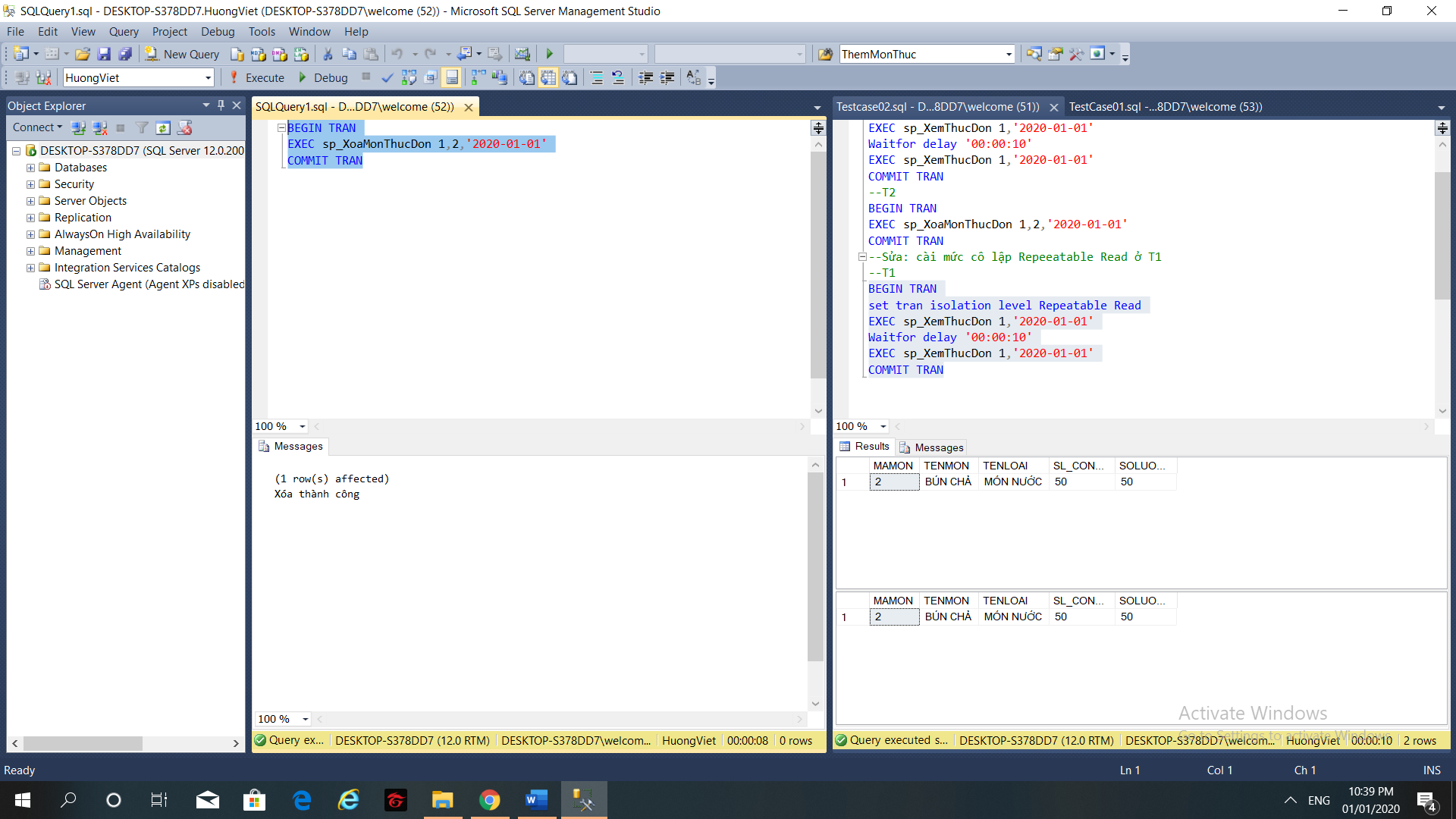
* Giao tác 1 tiến hành đọc dữ liệu của thực đơn và kết quả giống hệt dữ liệu ban đầu(chưa xóa).

Giao tác 1 đọc dữ liệu lần 2.

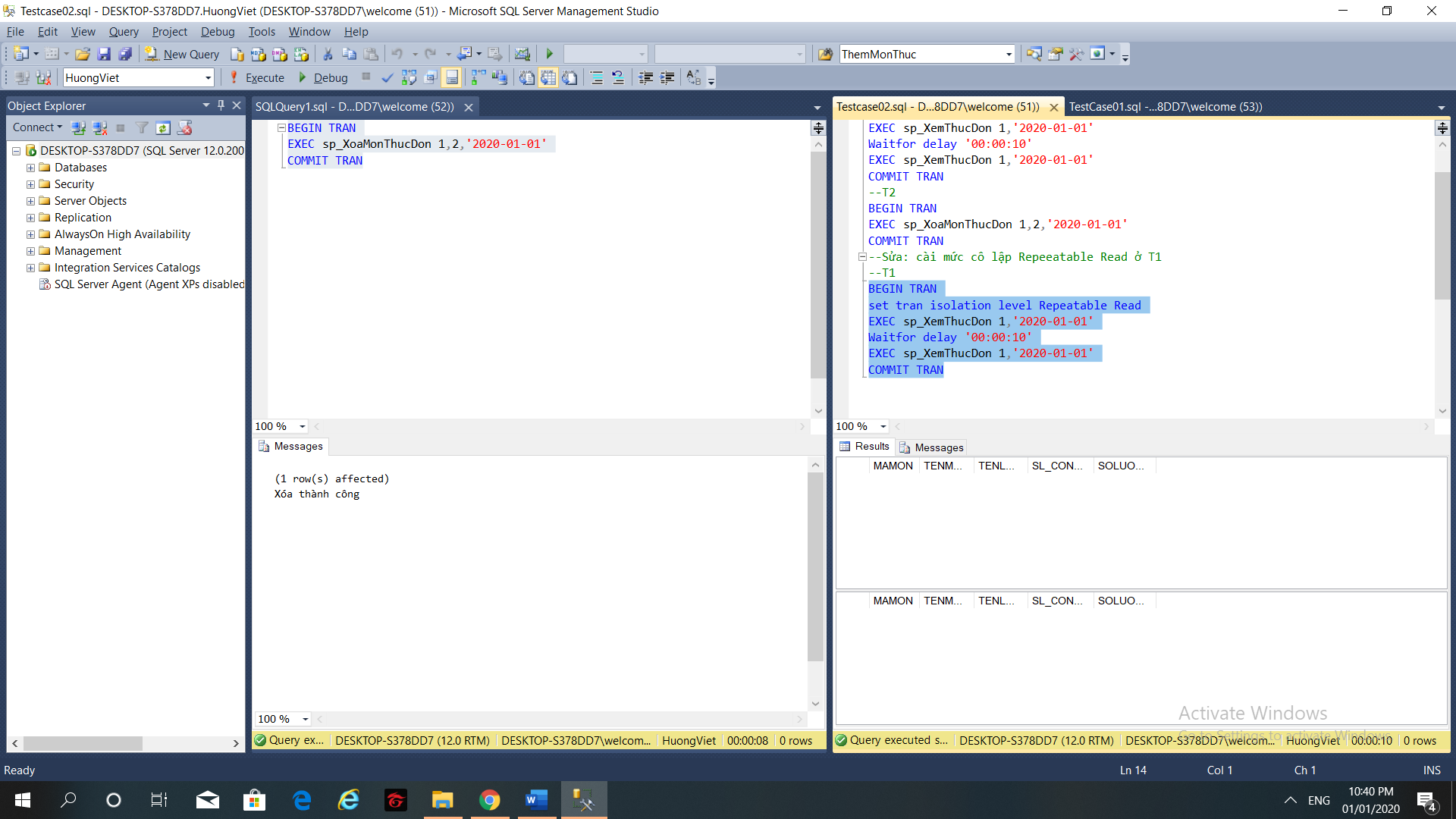
* Ở lần đọc thứ 2 này, Giao tác 1 đã thấy dữ liệu có sự thay đổi, cụ thể là dữ liệu đã bị xóa

1. Giải pháp: Set cơ chế khóa Repeatable Read

Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết giao tác 2, lúc này T2 không xóa được dữ liệu do T1 set mức cô lập Repeatable Read. Khóa S đã được giữ cho đến hết giao tác, T2 muốn xin khóa X thì phải đợi T1 hoàn thành xong giao tác.



Sau khi kết thúc giao tác T1, T2 mới vào thực hiện xóa dữ liệu, và đây là kết quả sau khi cập nhật:

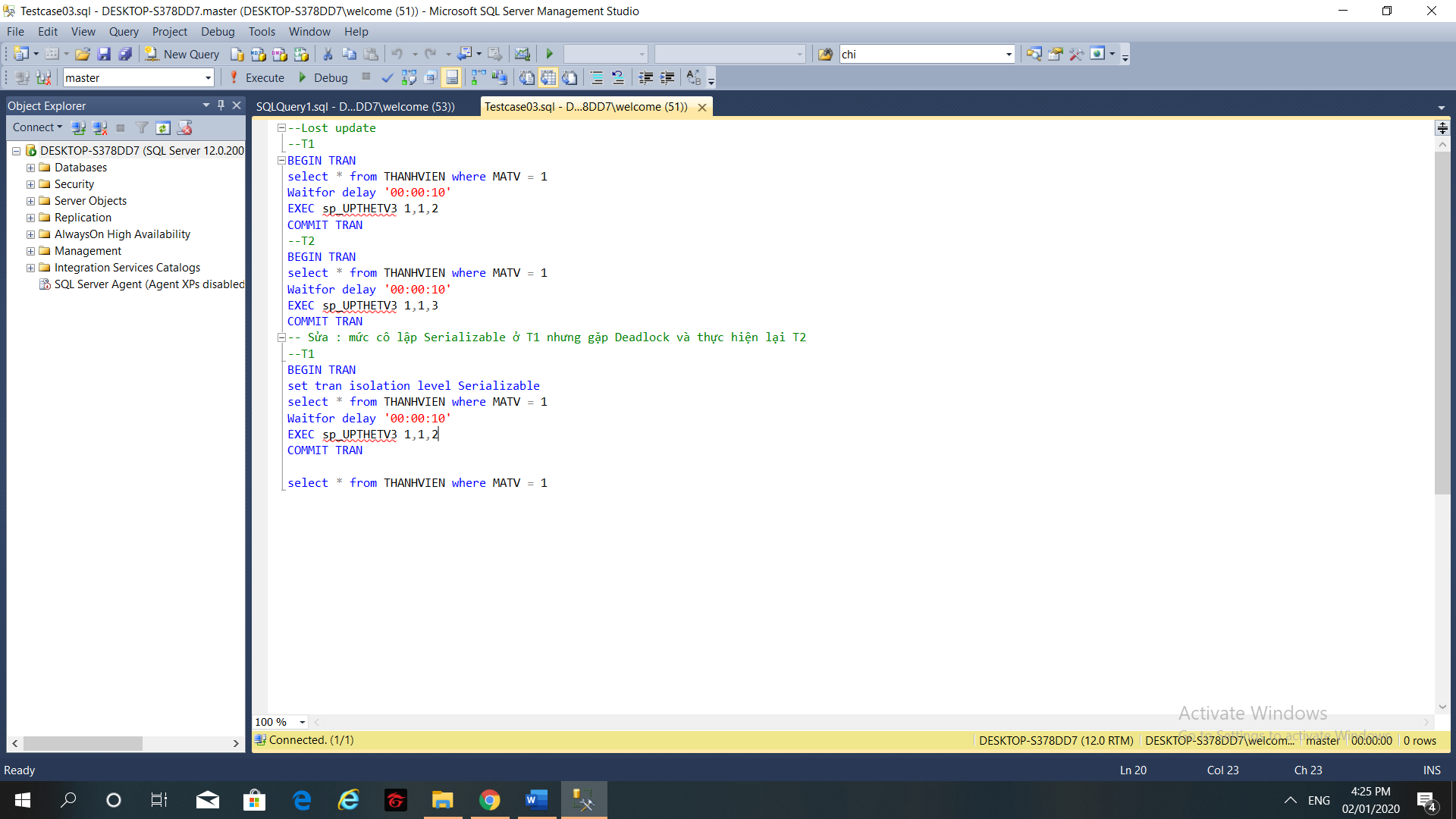


1. ***Lost update:***
2. ***Lỗi:*** T1 đăng nhập và thay đổi giá trị thẻ, sau đó T2 đăng nhập và cũng thay đổi giá trị thẻ. Qua đó update của T1 bị mất.

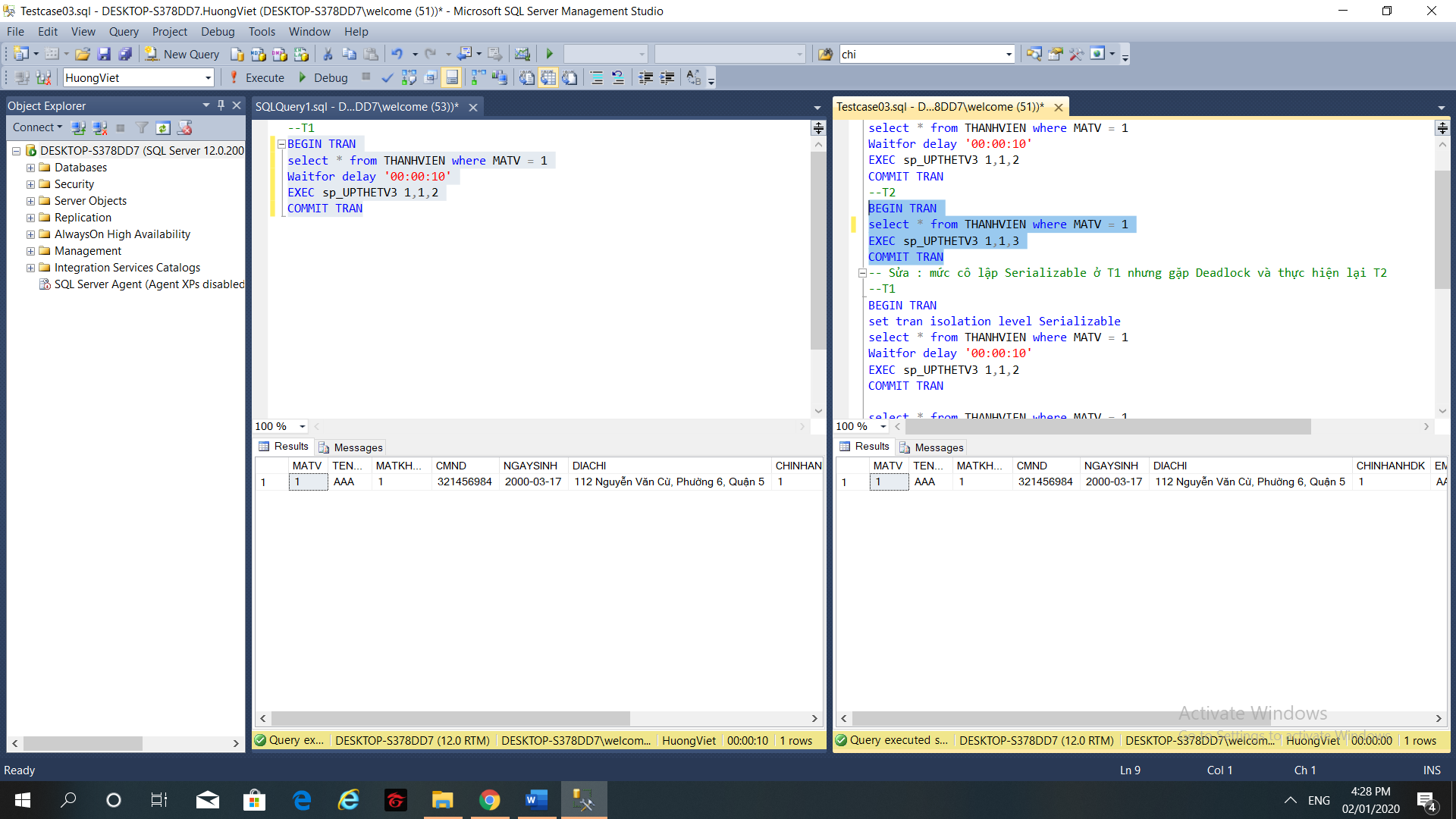
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC SP\_DangNhap 1,1  Waitfor delay '00:00:10'  EXEC sp\_UPTHETV3 1,1,2  COMMIT | BEGIN TRAN  EXEC SP\_DangNhap 1,1  Waitfor delay '00:00:10'  EXEC sp\_UPTHETV3 1,1,3  COMMIT |

1. Chạy lỗi trên SQL:

**Đoạn test case của trường hợp Lost Update: (Giả lập lấy cơ chế Read Committed)**



Chạy giao tác 1 , sau đó chạy hết tác 2 và ta có kết quả:

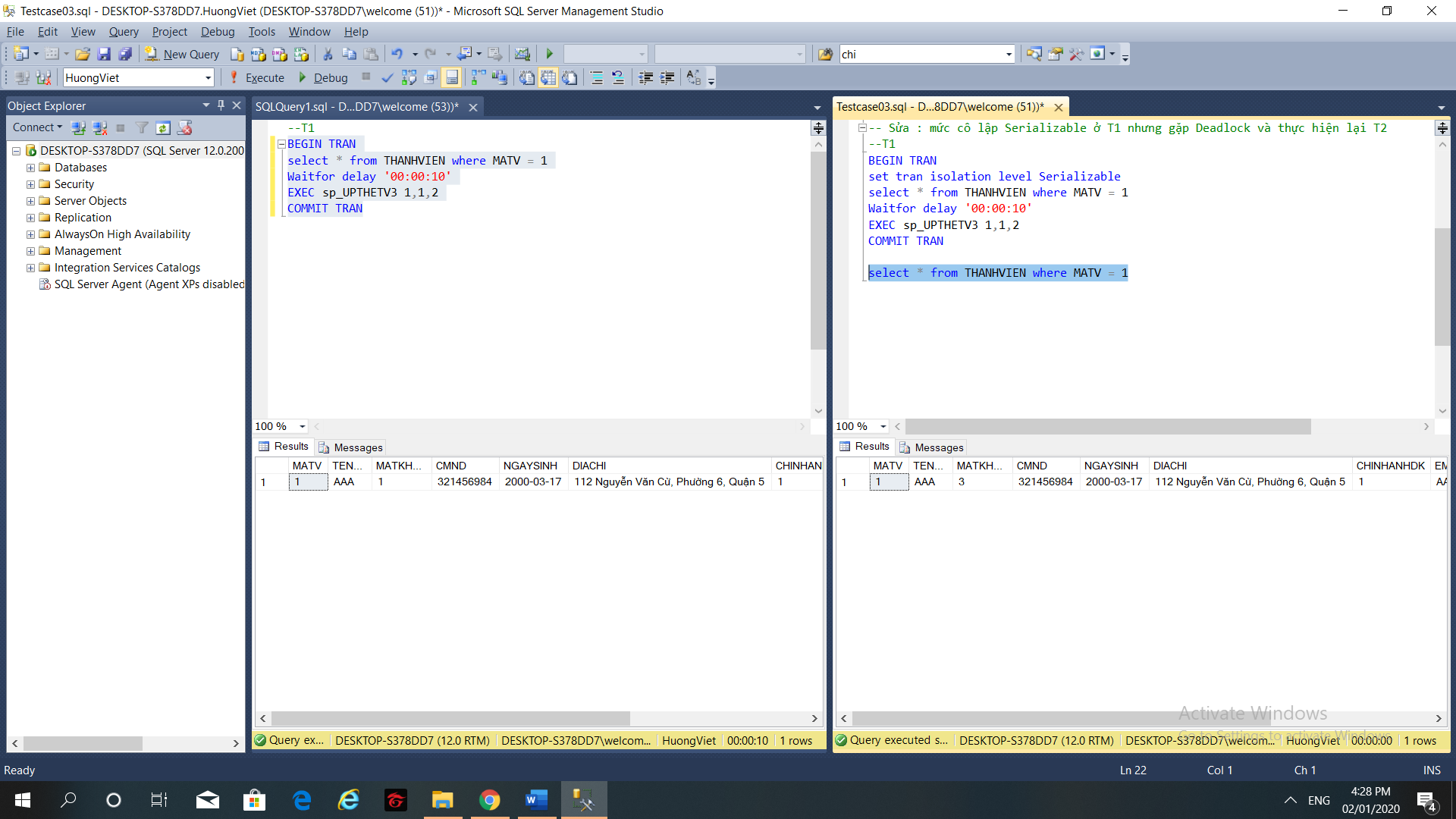


* 2 giao tác đều đọc được dữ liệu giống nhau, cũng là dữ liệu ban đầu.

Giao tác 1 cập nhật mật khẩu và comit:

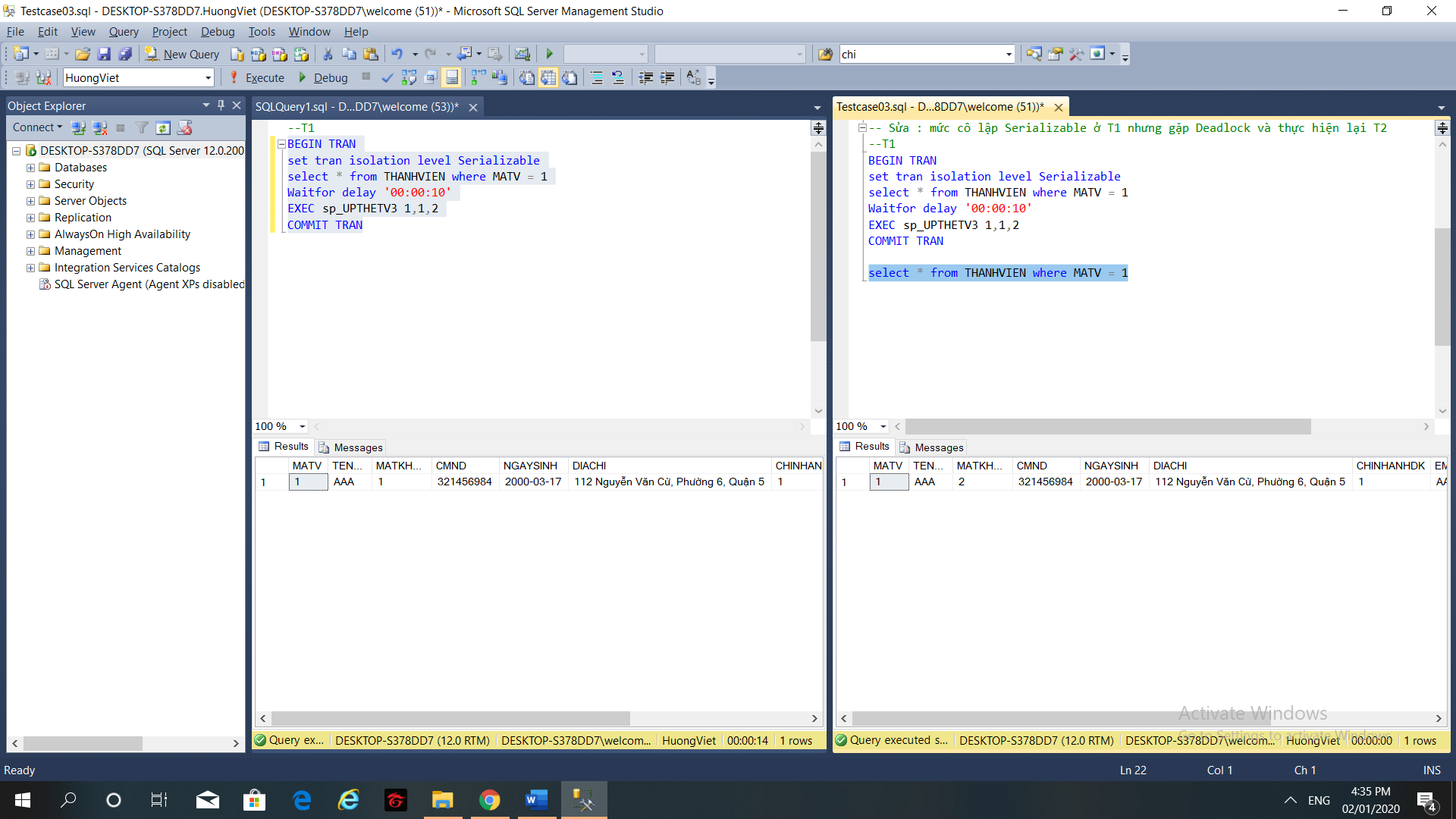
* Dữ liệu đã được cập nhật thành công

Giao tác 2 cập nhật địa chỉ và commit:



* Giao tác 2 đã cập nhật dữ liệu đè lên giao tác 1, nếu người dùng của giao tác 1 vào xem lại dữ liệu sẽ thấy có sự thay đổi. Ở đây là pass = 3 không là pass = 2 như T1 mong muốn.

1. ***Giải pháp:*** Set cơ chế khóa Serializable.



Pass của thành viên có mã là 1 đã thành 2 không còn là 3 như trước khi set khóa Serializable.

1. ***Phantom:***
2. ***Lỗi:*** Gỉa sử T1 thống kê vào ngày hôm nay tại chi nhánh 1. Sau đó T2 insert vào bảng đơn hàng. T1 thống kê lại thì thấy thêm dữ liệu T2 vừa thêm vào dù chưa commit. Lỗi phantom.

|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_ThongKeTheoNgayCN 1  Waitfor delay '00:00:10'  EXEC sp\_ThongKeTheoNgayCN 1  COMMIT TRAN | BEGIN TRAN  EXEC sp\_InDH null,null,1,N'Trực tiếp',N'Qua thẻ'  EXEC sp\_InCTDH 1,1,1  COMMIT TRAN |

1. Chạy lỗi trên SQL:

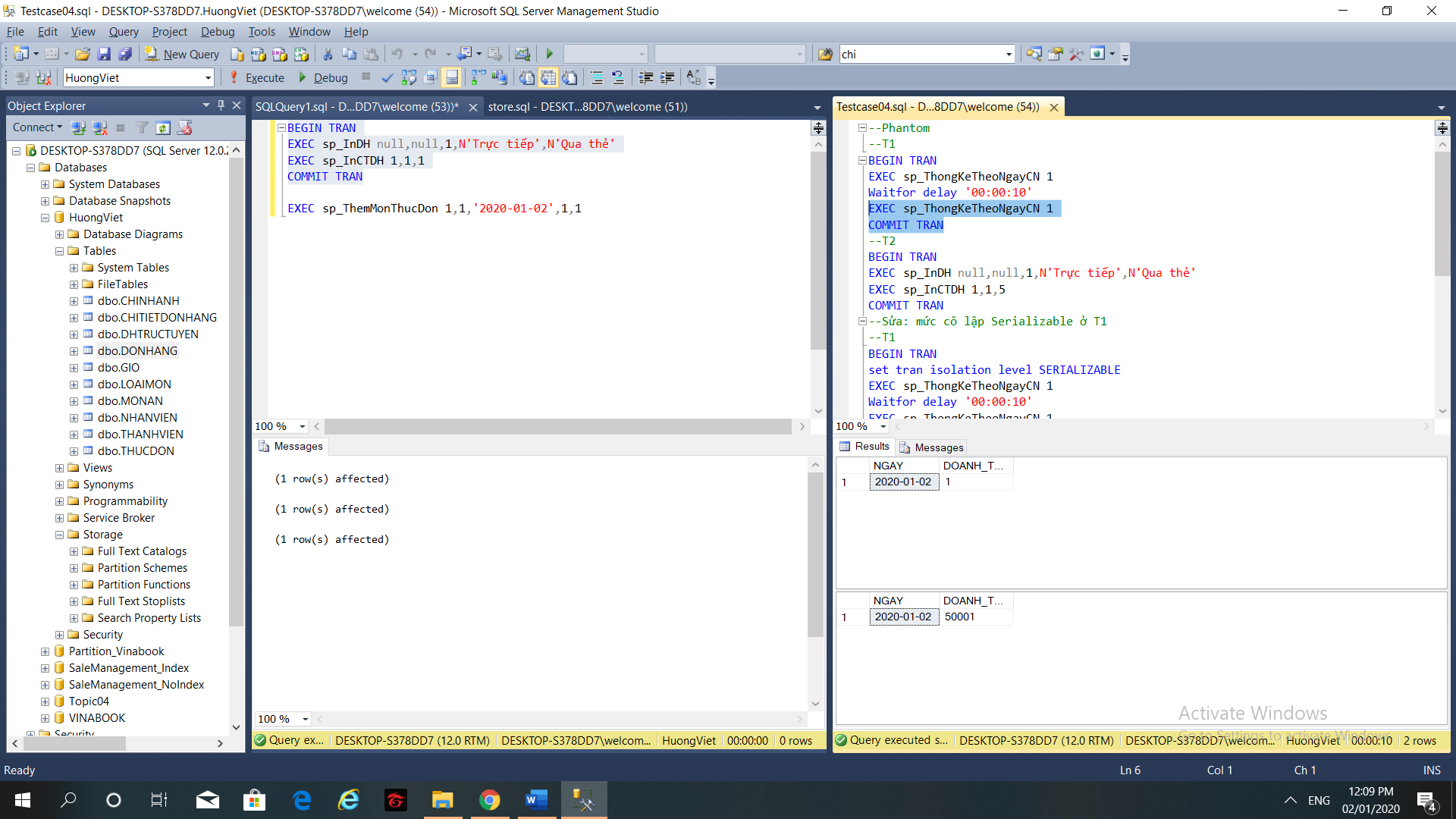
**Đoạn test case của trường hợp Phantom: (Giả lập lấy cơ chế Read Committed)**



Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:

* Giao tác 1 xem dữ liệu của chi nhánh 2 và nhận được dữ liệu ban đầu (không cập nhật/ thêm). Giao tác 2 đã thêm dữ liệu thành công.

Giao tác 1 tiến hành xem dữ liệu của chi nhánh 1 một lần nữa:



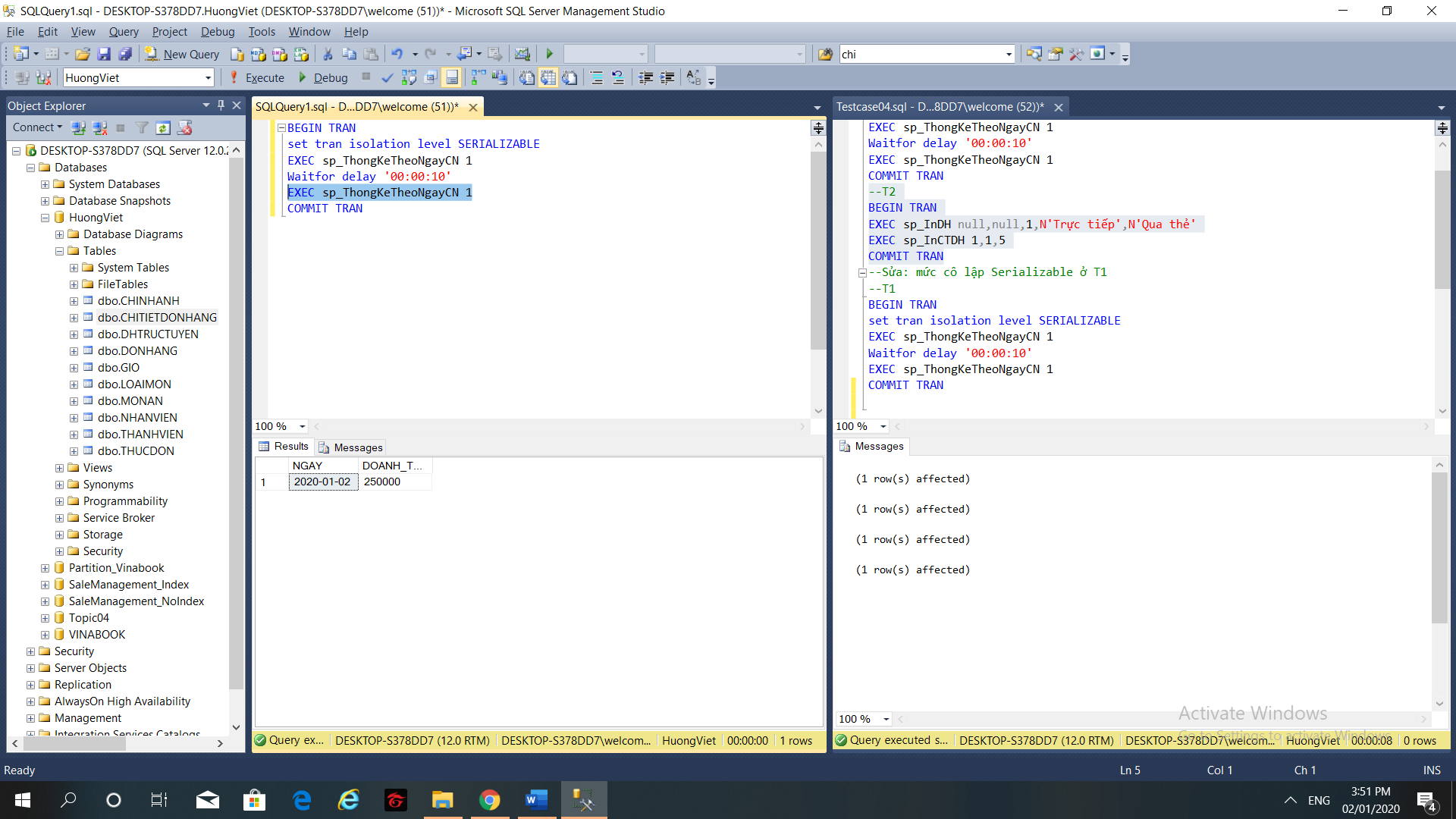
* Giao tác 1 đọc dữ liệu và thấy kết quả khác.

1. ***Giải pháp:*** Set cơ chế khóa Serializable

Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết giao tác 2, lúc này T2 không thêm được dữ liệu do T1 set mức cô lập Serializable. Khóa S đã được giữ cho đến hết giao tác, không cho insert dòng thỏa điều kiện thiết lập khóa, nên T2 muốn thêm dữ liệu thì phải đợi T1 commit.



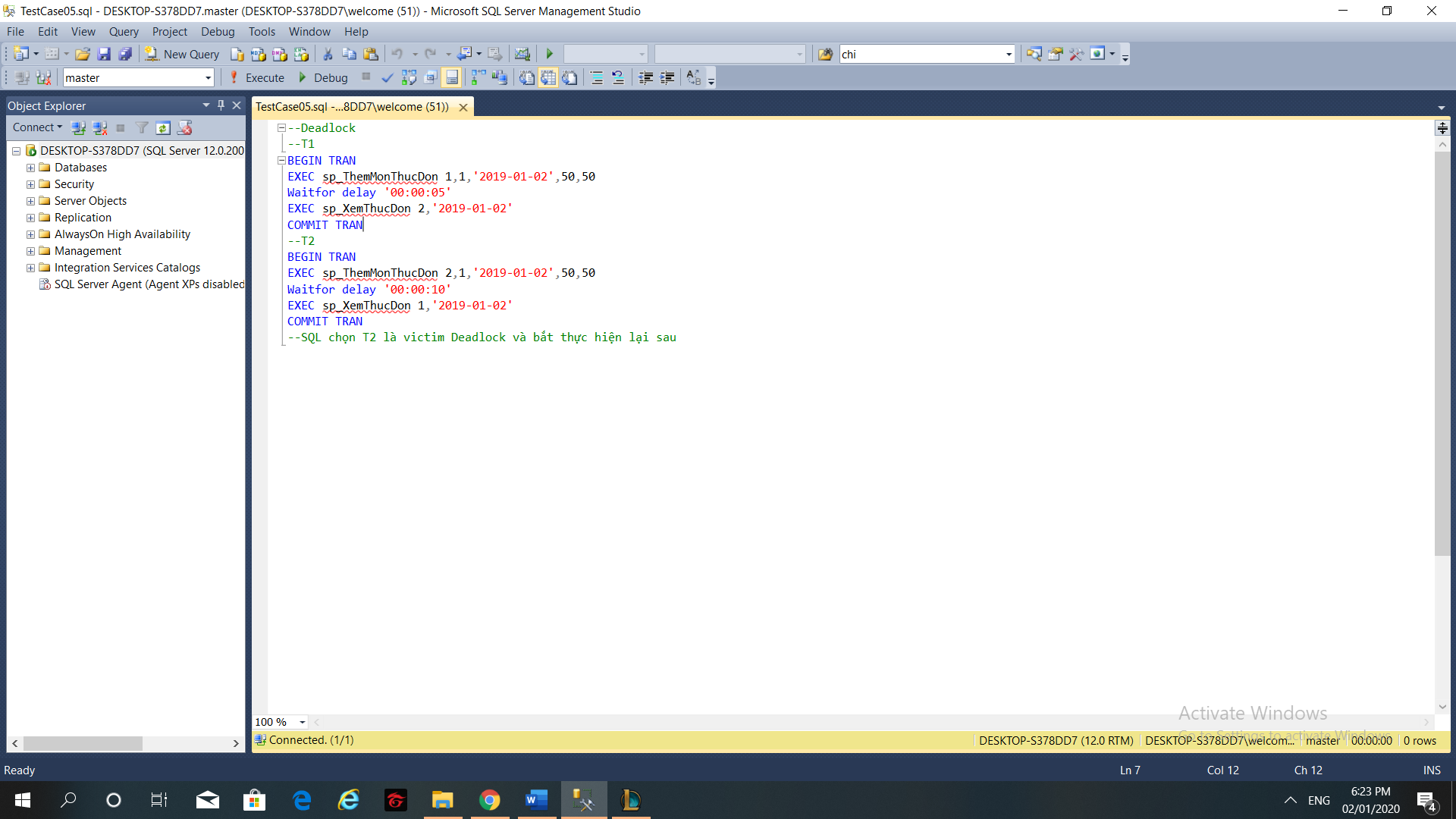
Đây là kết quả sau khi T1 commit:



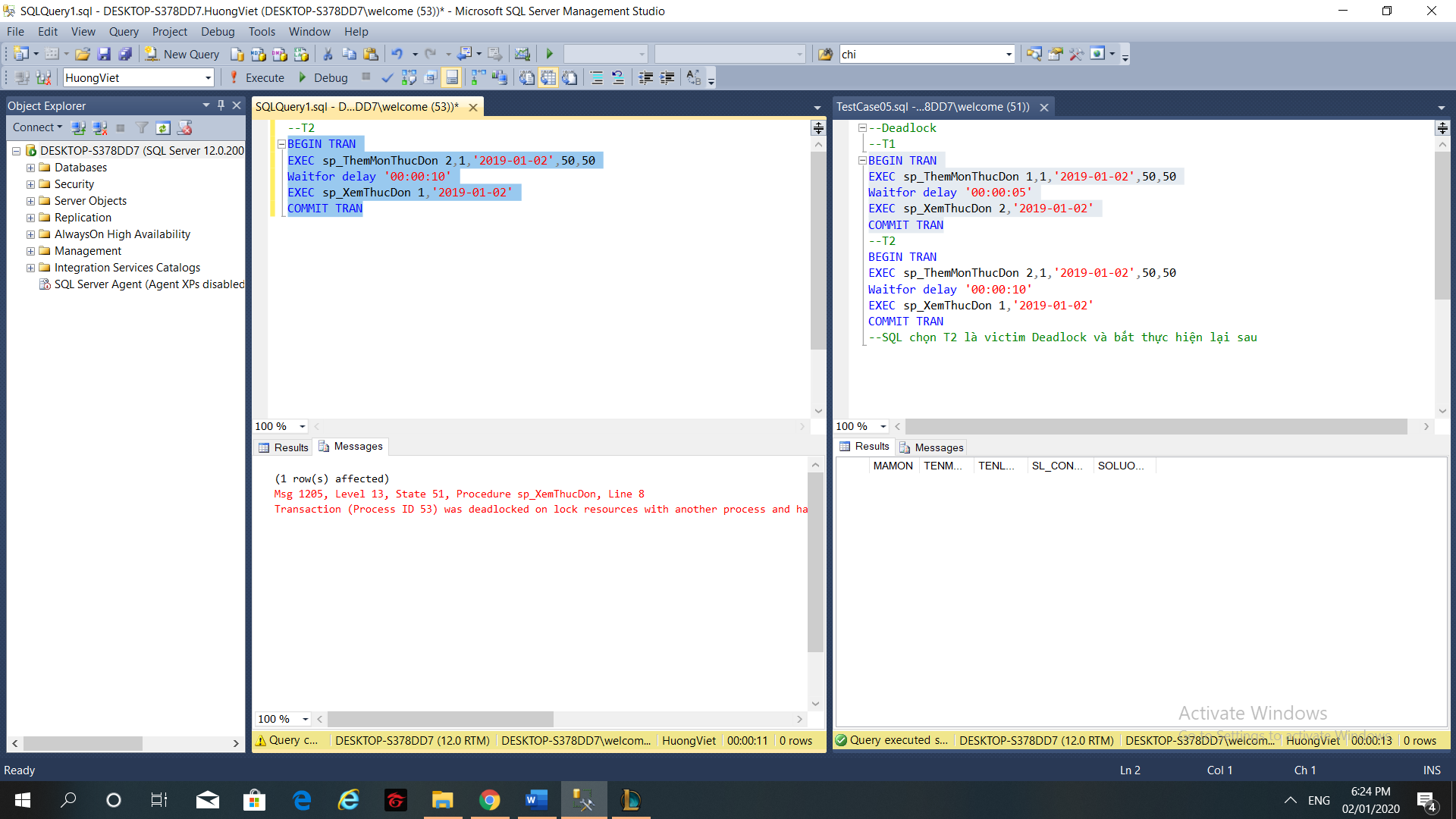
T2 thêm dữ liệu thành công.

1. ***DeadLock***

**Đoạn test case của trường hợp Deadlock: (Giả lập lấy cơ chế Read Committed)**



Chạy T1 trước rồi chạy T2.



T1 được chạy trước, T2 bị SQL SERVER hủy và chạy sau T1 vì chương trình bị DeadLock.