|  |  |  |  |
| --- | --- | --- | --- |
| Tên thủ tục | Mục đích | Input | Output |
| sp\_THEMTHANHVIENonl | Khách hàng đăng ký thẻ thành viên | @TENTV, @MATKHAU, @CMND,NGSINH , @CHINHANHDK, @DIACHI, @EMAIL | Dữ liệu mới được thêm vào bảng THANHVIEN |
| SP\_DangNhap | Thành viên đăng nhập vào hệ thống | @user, @pass | Thông tin của thành viên |
| sp\_Khach | Nhân viên lễ tân thêm thông tin của khách hàng để làm thẻ thành viên | @tentv, @cmnd, @ngay\_sinh , @chinhanhdk, @DIACHI, @email, @pass | Dữ liệu mới được cập nhật vào bảng THANHVIEN |
| sp\_SuaKhach | Thành viên cập nhật thông tin của mình | @matv, @tentv, @cmnd, @ngay\_sinh, @dia\_chi,@chinhanhdk, , @email | Dữ liệu mới được cập nhật vào bảng THANHVIEN |
| sp\_XoaKhach | Xóa thành viên | @matv | Dữ liệu của thành viên sẽ bị xóa khỏi bảng THANHVIEN |
| sp\_UPTHETV3 | Nhân viên xóa tài khoản thành viên | @MATV, @MATKHAU, @MATKHAUMOI | Cập nhật mật khẩu mới cho thành viên |

* ***Chức năng (thủ tục chạy):***
* ***Thủ tục tranh chấp:***

|  |  |  |
| --- | --- | --- |
| Tên lỗi | Kịch bản lỗi | Xử lý |
| Dirty Read | Giả sửkhách hàng số 1 vào xem và thấy số lượng còn lại của món số 1 tại chi nhánh 1 là 1, người này tiến hành đặt hàng nhưng chưa commit thao tác.  Trong lúc này thì khách hàng số 2 cùng vào xem món 1 tại chi nhánh 1 và thấy số lượng còn lại là 0, người này không đặt nữa. Khách 1 không đặt món nữa và rollback lại, lúc này khách hàng số 2 đã  đọc dữ liệu sai.  Ở ví dụ này, để xảy ra lỗi thì giả lập mức cô lập cho giao tác T2 là Read Uncommitted | Thiết lập mức cô lập Read Committed |
| Unrepeatable Read | Khách hàng có mã thành viên = 1 đang xem thông tin tài khoản của mình ở chi nhánh 1, thì có người khác cũng vào tài khoản này và sửa thông tin DIACHI.  Khách hàng thực hiện xem thông tin lại một lần nữa thì thông tin ban đầu đã bị thay đổi. | Thiết lập mức cô lập Repeatable Read |
| Lost Update | Giả sử có 2 người đồng thời truy cập vào tài khoản thành viên 1. Người số 1 vào cập nhật lại địa chỉ là ‘số 7,Nguyễn Hữu Cảnh, phường 3, quận 2, TPHCM’, người số 2 vào cũng cập nhật lại địa chỉ ‘122, Nguyễn Văn Cừ, phường 10, quận 5, TPHCM’, người thứ 1 nếu lần sau vào xem lại thông tin thì sẽ không giống với thông tin ban đầu mình đã sửa nữa. | Thiết lập mức cô lập Serializable |
| Phantom | Giả sử nhân viên quản lý khách hàng đang xem lại thông tin của tất cả khách hàng đã đăng ký thành viên trong chi nhánh 2.  Trong lúc này, một khách hàng khác vào và đăng ký thành viên trong chi nhánh 2 bằng kênh online. Khi nhân viên xem lại thì thấy số lượng khách hàng đăng ký thành viên trong chi nhánh 2 bị thay đổi. | Thiết lập mức cô lập Serializable |
| Deadlock | Giả sử có 2 người đồng thời truy cập vào tài khoản thành viên 1. Người số 1 vào cập nhật lại địa chỉ là ‘số 7,Nguyễn Hữu Cảnh, phường 3, quận 2, TPHCM’, người số 2 vào cũng cập nhật lại địa chỉ ‘122, Nguyễn Văn Cừ, phường 10, quận 5, TPHCM’, người thứ 1 nếu lần sau vào xem lại thông tin thì sẽ không giống với thông tin ban đầu mình đã sửa nữa. | Thiết lập mức cô lập Serializable |

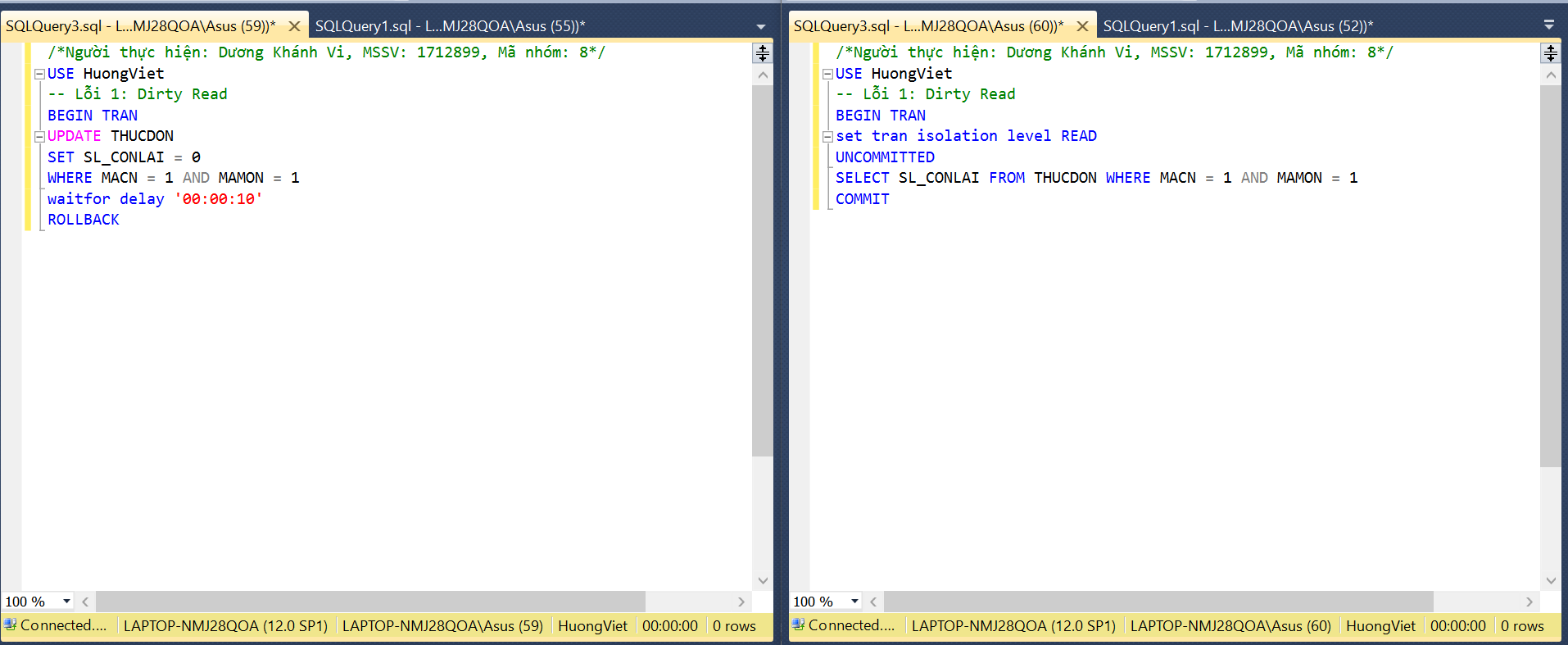
* ***Chi tiết thủ tục tranh chấp:***

1. ***Dirty read:***
2. ***Lỗi :*** Giả sửkhách hàng số 1 vào xem và thấy số lượng còn lại của món số 1 tại chi nhánh 1 là 1, người này tiến hành đặt hàng nhưng chưa commit thao tác. Trong lúc này thì khách hàng số 2 cùng vào xem món 1 tại chi nhánh 1 và thấy số lượng còn lại là 0, người này không đặt nữa. Khách 1 không đặt món nữa và rollback lại, lúc này khách hàng số 2 đã đọc dữ liệu sai. Ở ví dụ này, để xảy ra lỗi thì giả lập mức cô lập cho giao tác T2 là Read Uncommitted

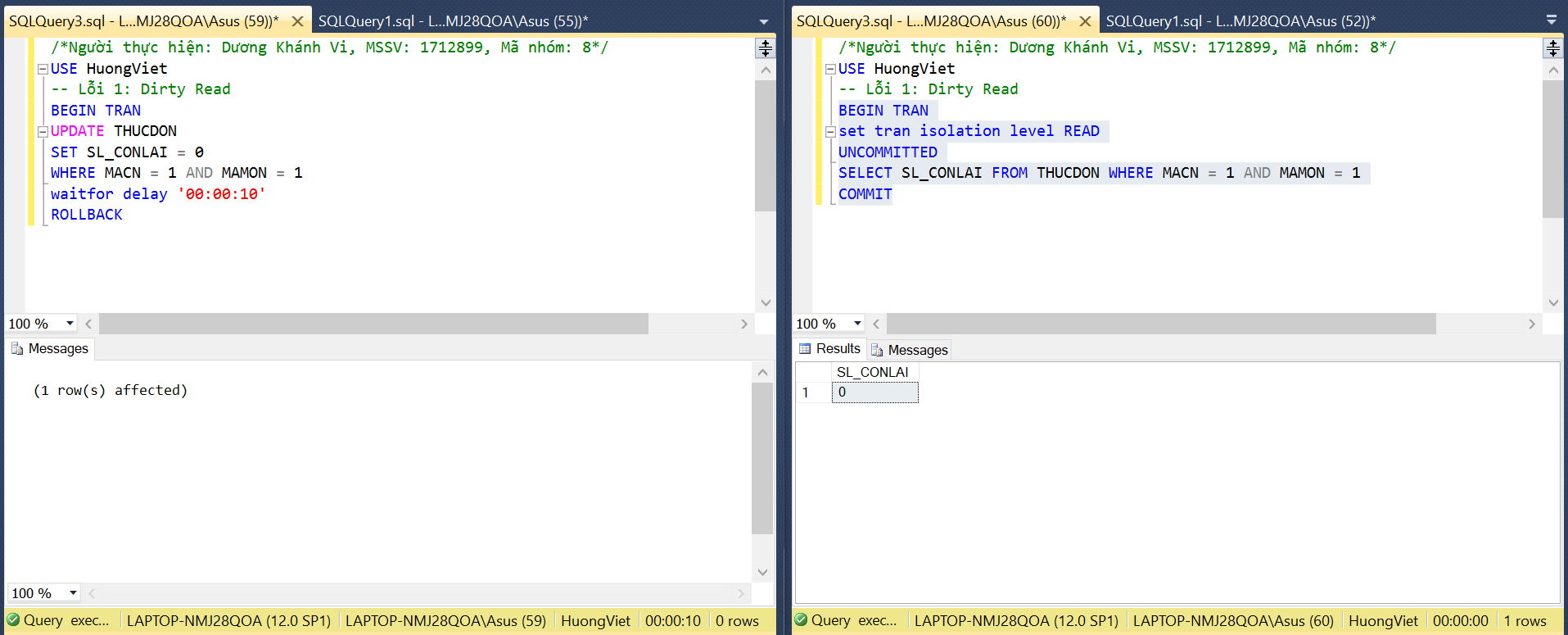
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  UPDATE THUCDON  SET SL\_CONLAI = 0  WHERE MACN = 1 AND MAMON = 1  waitfor delay‘00:00:10’  ROLLBACK | BEGIN TRAN  set tran isolation level READ UNCOMMITTED  SELECT SL\_CONLAI  FROM THUCDON WHERE MATD = 1 AND MACN = 1  COMMIT |

1. Chạy lỗi trên SQL:

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

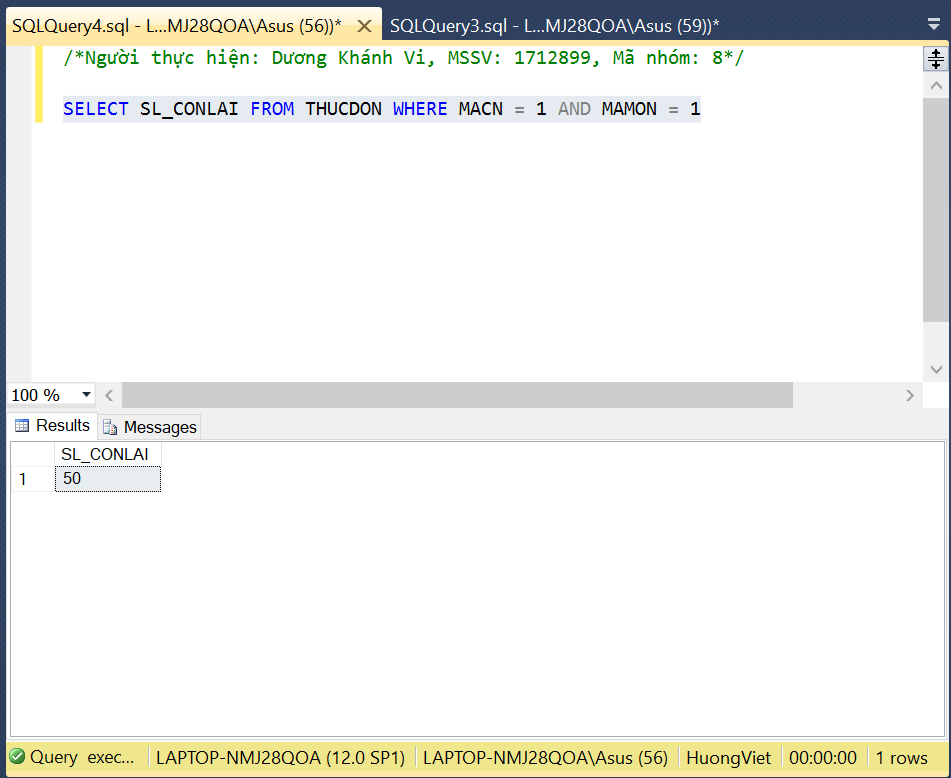


Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:

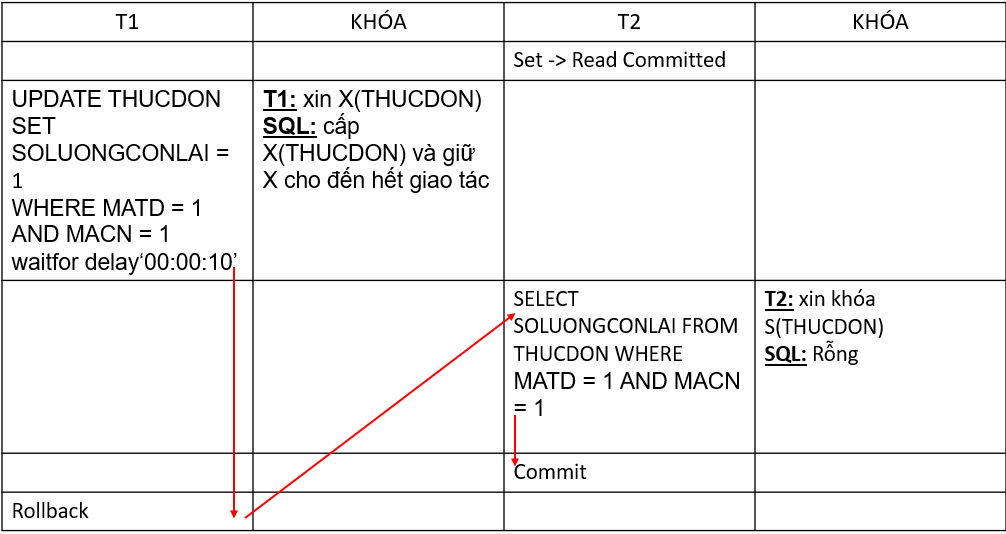


* Giao tác 2 đã đọc dữ liệu mà giao tác 1 đã cập nhật nhưng chưa rollback với nội dung Số lượng còn lại của món ăn có mã 1 là 0.

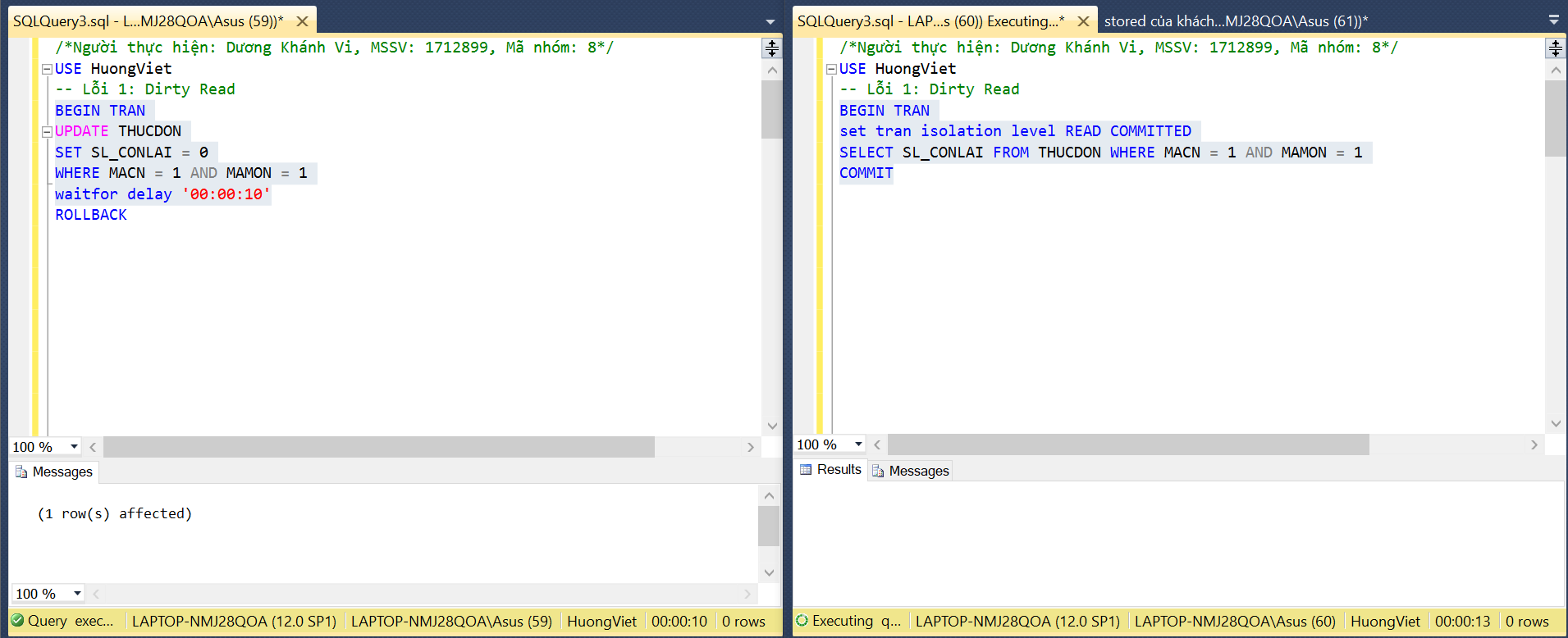
Sau khi Giao tác 1 rollback, thử truy vấn lại thì số lượng còn lại đã trở lại 50.



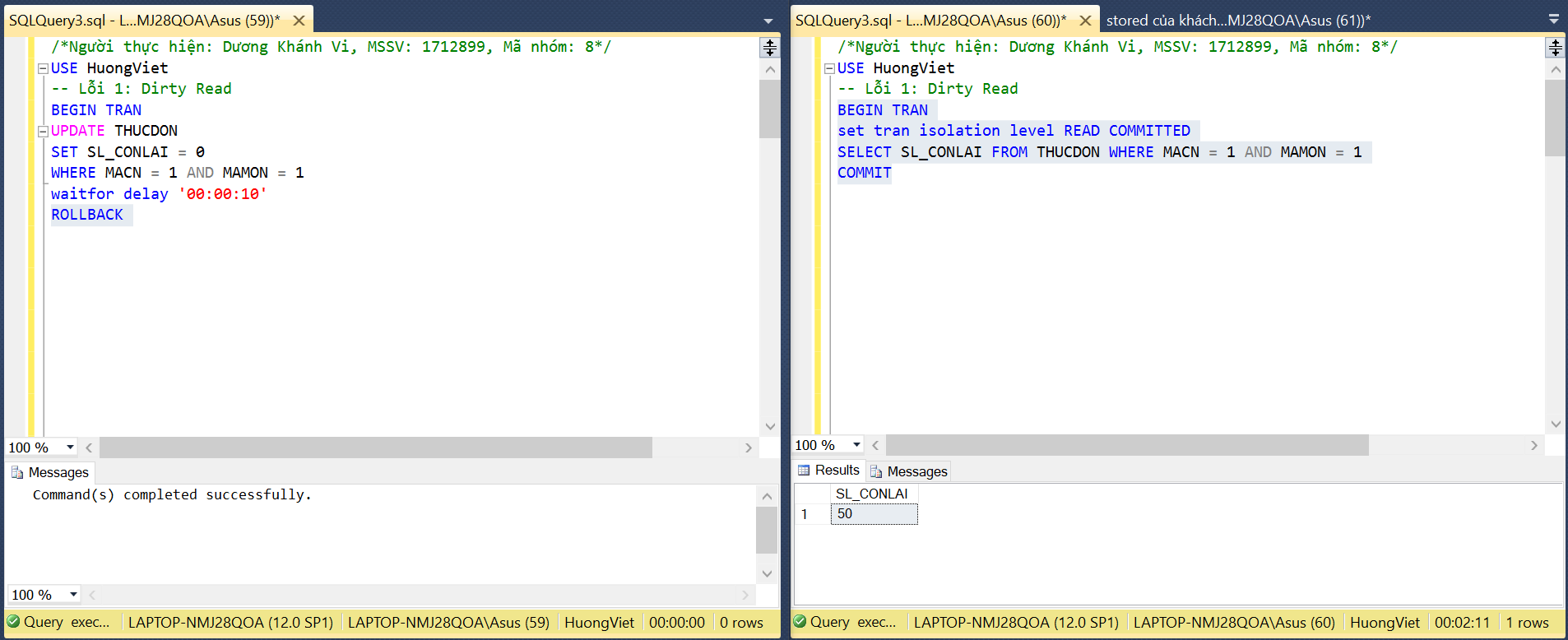
1. ***Giải pháp:*** Set cơ chế khóa Read Committed



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.

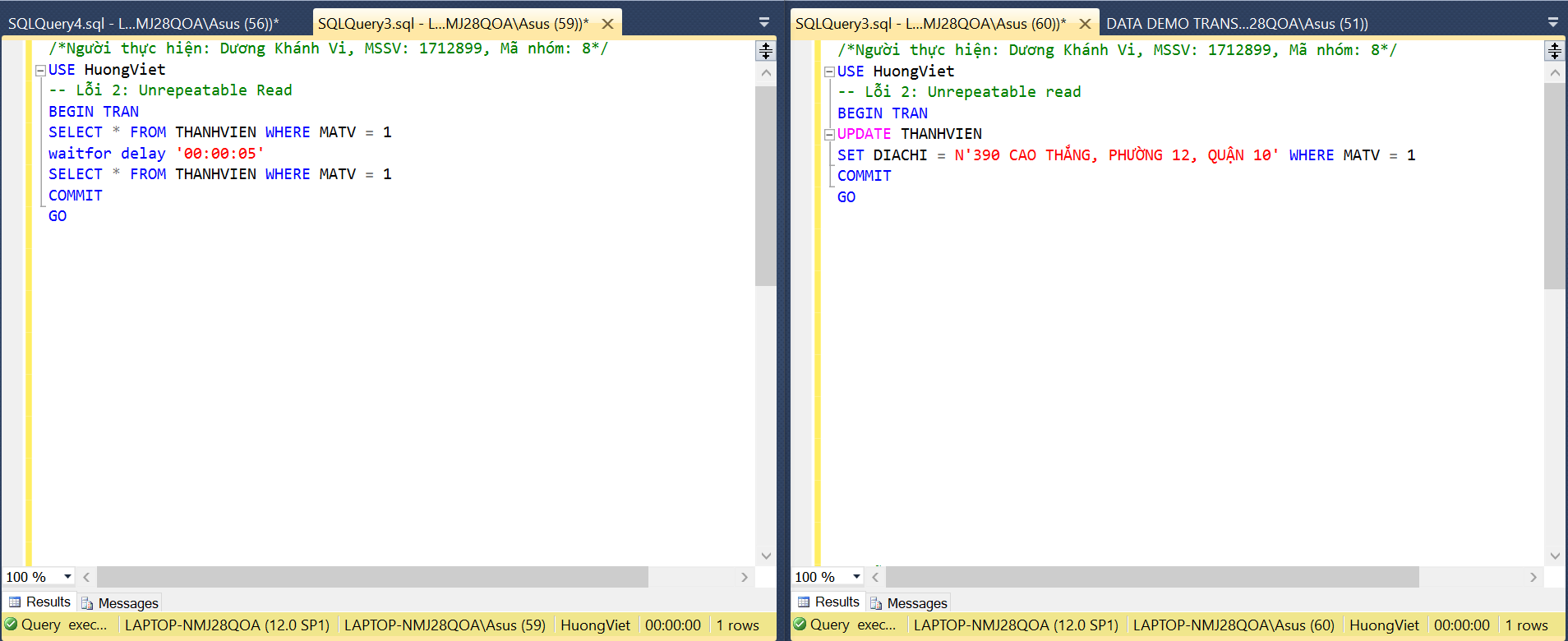


1. ***Unrepeatable read:***
2. ***Lỗi 1:*** Khách hàng có mã thành viên = 1 đang xem thông tin tài khoản của mình ở chi nhánh 1, thì có người khác cũng vào tài khoản này và sửa thông tin DIACHI .Khách hàng thực hiện xem thông tin lại một lần nữa thì thông tin ban đầu đã bị thay đổi.

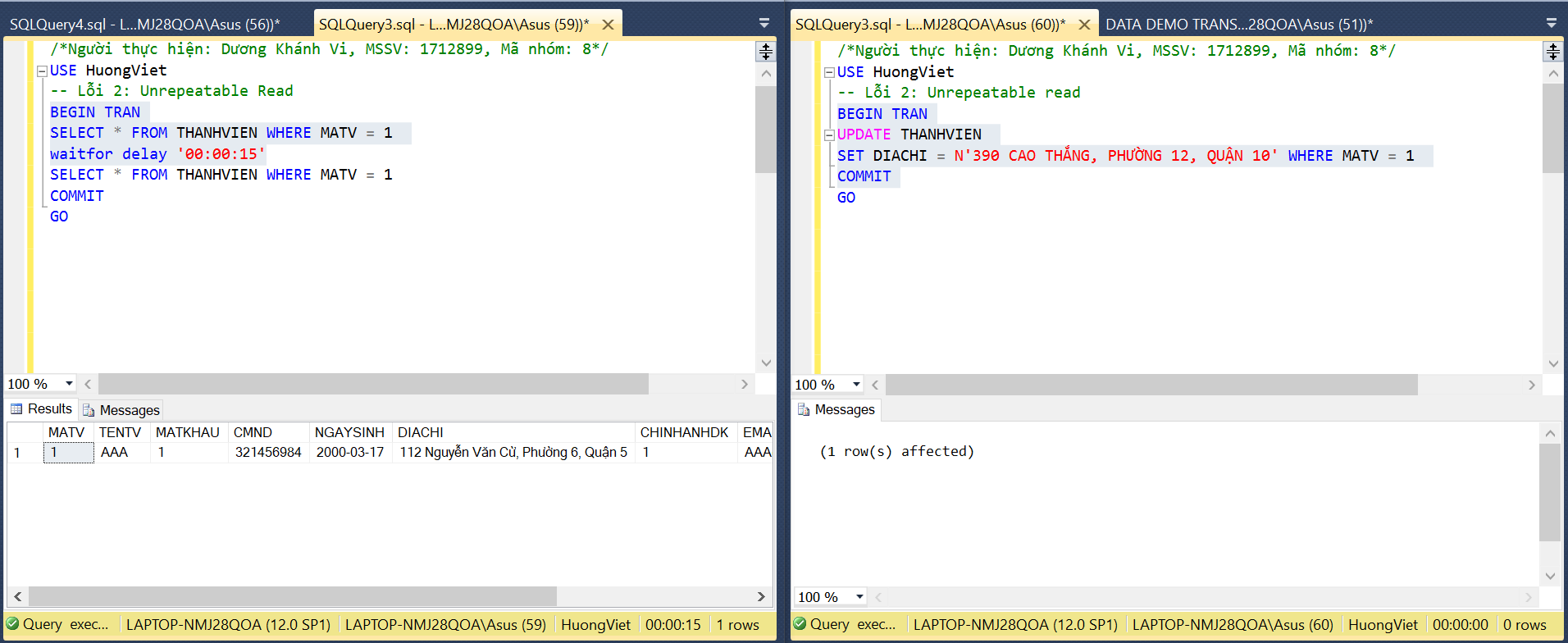
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  SELECT \* FROM THANHVIEN WHERE MATV = 1  waitfor delay‘00:00:10’  SELECT \* FROM THANHVIEN WHERE MATV = 1  COMMIT | BEGIN TRAN  UPDATE THANHVIEN  SET DIACHI = N’390 CAO THẮNG, PHƯỜNG 12, QUẬN 10’ WHERE MATV = 1  COMMIT |

1. Chạy lỗi trên SQL

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

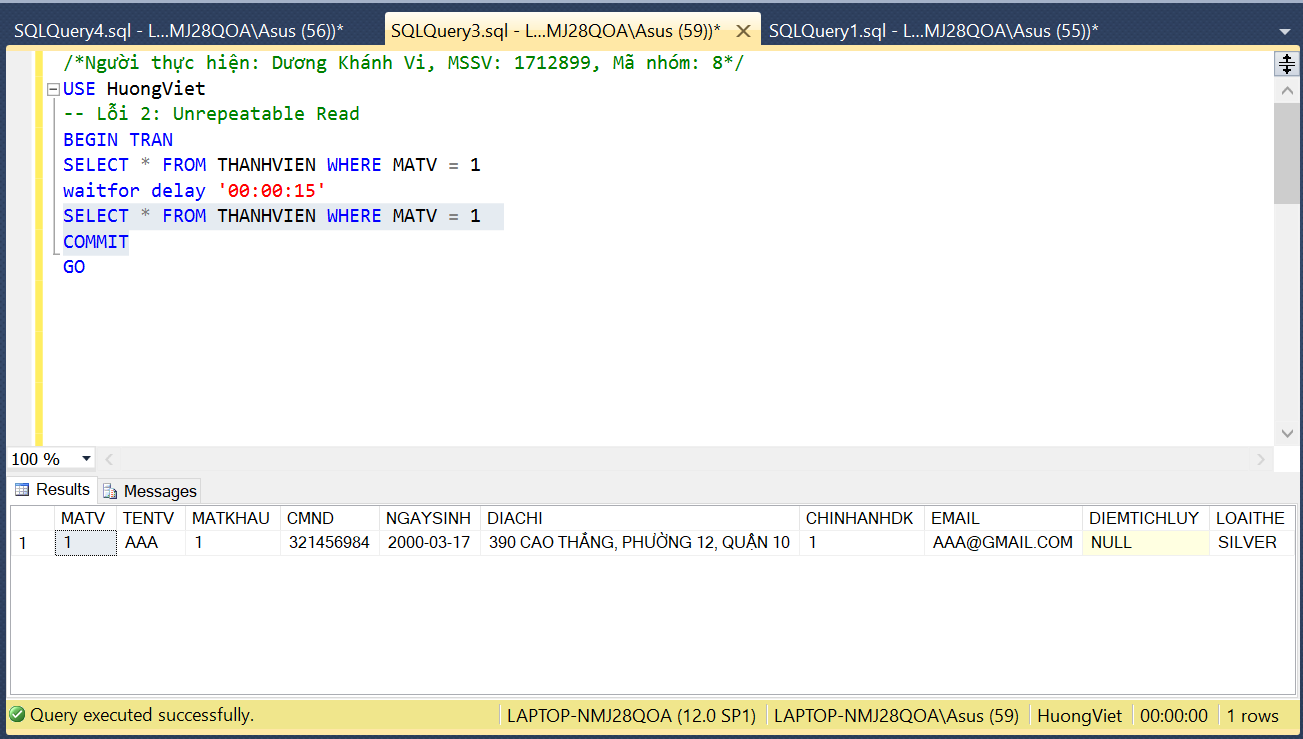


Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:



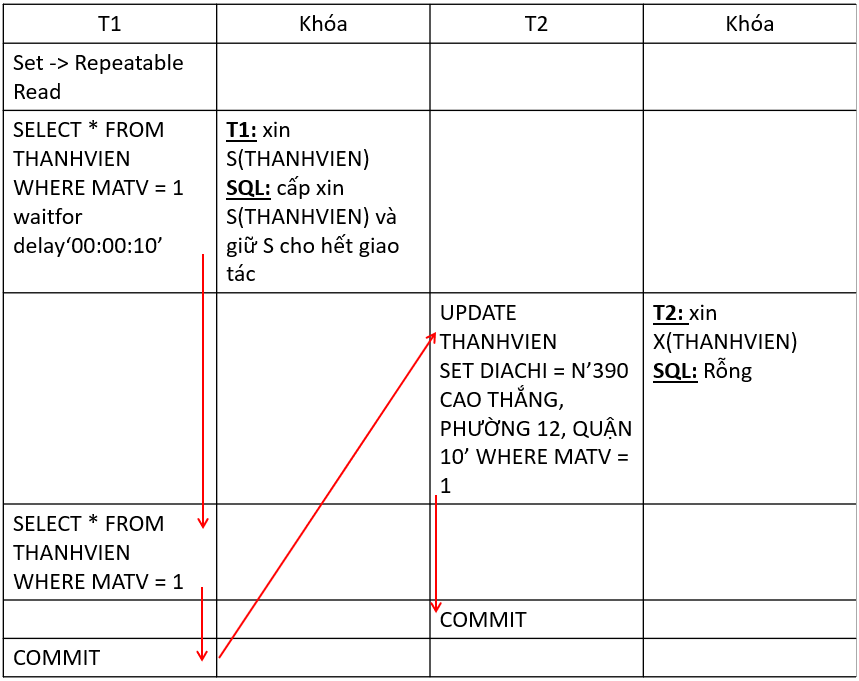
* Giao tác 1 tiến hành đọc dữ liệu của mã tv = 1 và kết quả giống hệt dữ liệu ban đầu(chưa chỉnh sửa).

Giao tác 1 đọc dữ liệu của mã tv = 1 lần 2:

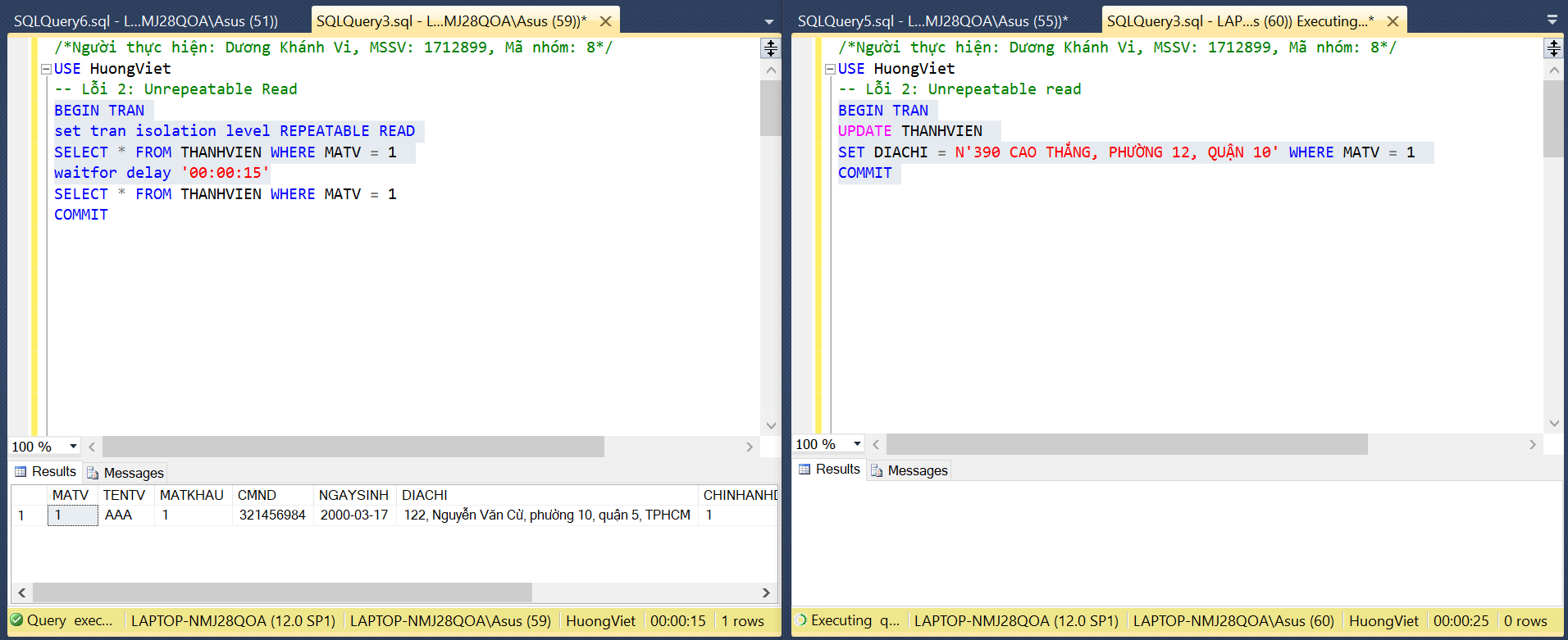


* Ở lần đọc thứ 2 này, Giao tác 1 đã thấy dữ liệu có sự thay đổi, cụ thể là địa chỉ đã bị thay đổi.

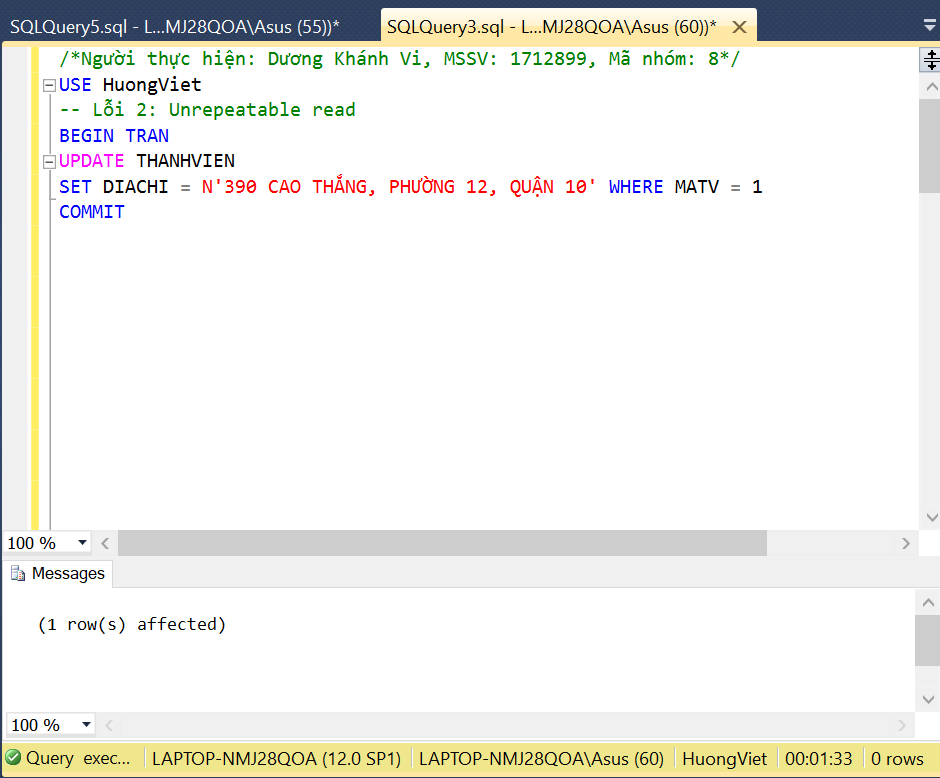
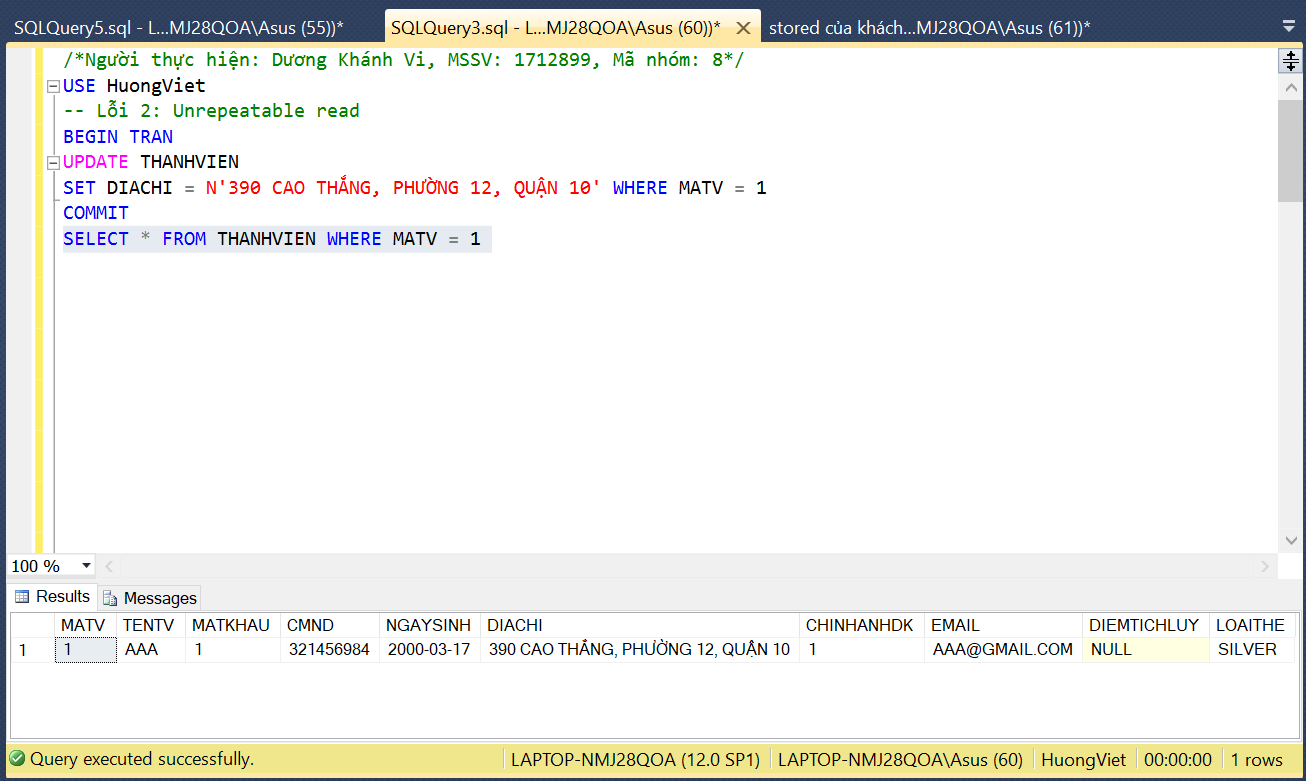
1. Giải pháp: Set cơ chế khóa Repeatable Read



Chạy giao tác 1 đến lệnh waitfor delay ’00:00:15’, sau đó chạy hết giao tác 2, lúc này T2 không cập nhật 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 cập nhật dữ liệu của MATV = 1, và đây là kết quả sau khi cập nhật:

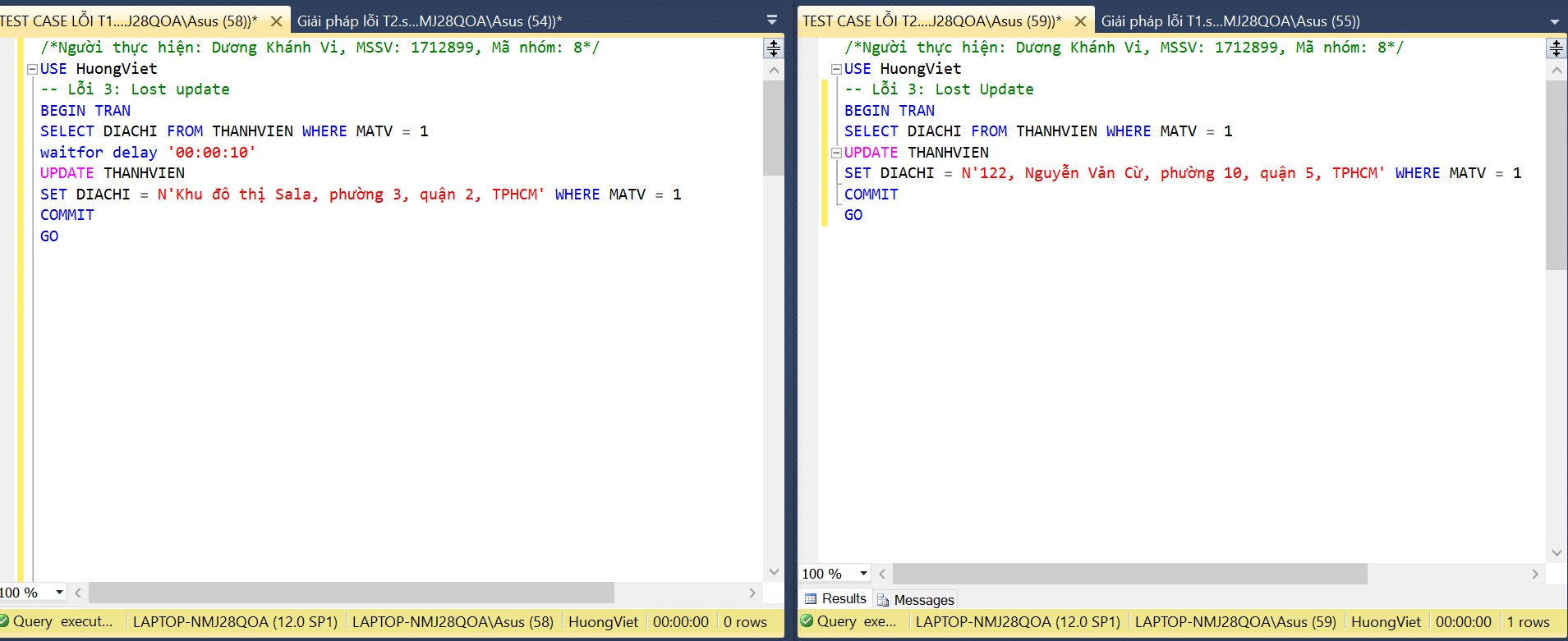
 

1. ***Lost update:***
2. ***Lỗi:*** Giả sử có 2 người đồng thời truy cập vào tài khoản thành viên 1. Người số 1 vào cập nhật lại địa chỉ là ‘số 7,Nguyễn Hữu Cảnh, phường 3, quận 2, TPHCM’, người số 2 vào cũng cập nhật lại địa chỉ ‘122, Nguyễn Văn Cừ, phường 10, quận 5, TPHCM’, người thứ 1 nếu lần sau vào xem lại thông tin thì sẽ không giống với thông tin ban đầu mình đã sửa nữa.

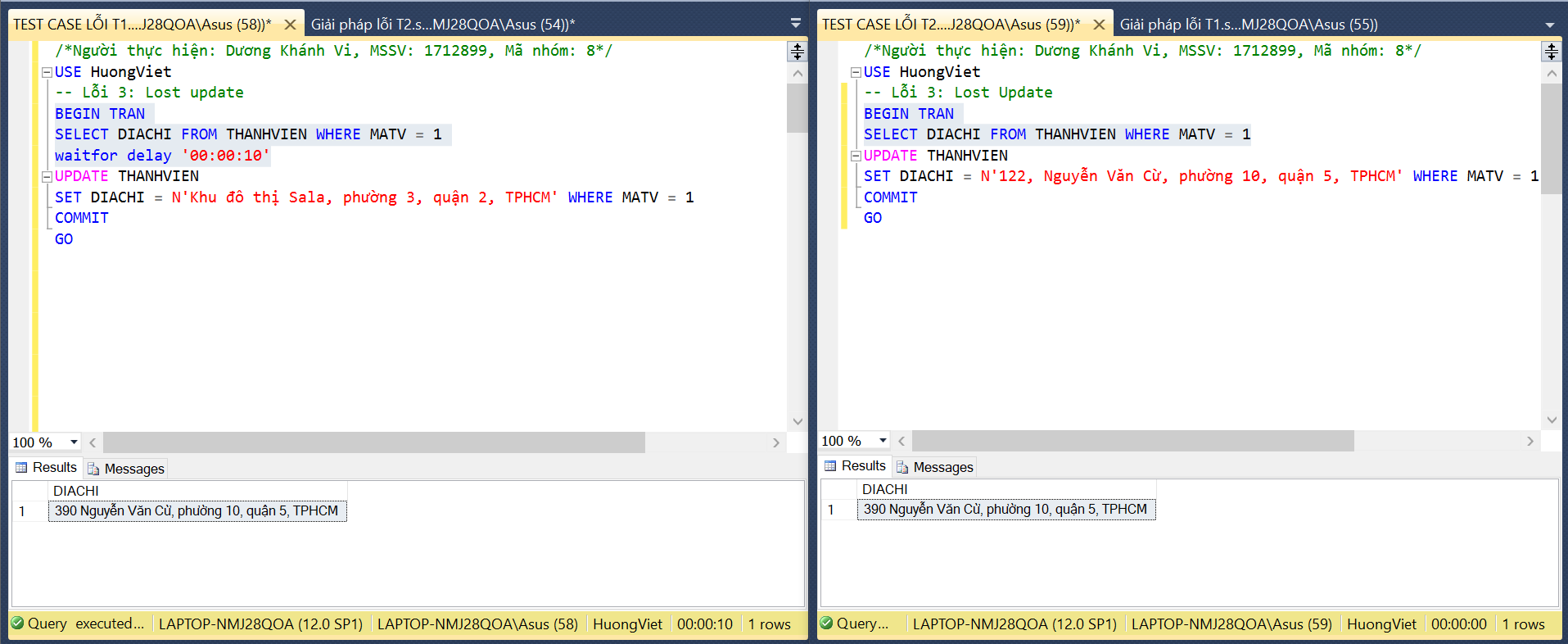
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  SELECT DIACHI FROM THANHVIEN WHERE MATV = 1  waitfor delay‘00:00:10’  UPDATE THANHVIEN  SET DIACHI = N ‘Khu đô thị Sala, phường 3, quận 2, TPHCM’  COMMIT | BEGIN TRAN  SELECT DIACHI FROM THANHVIEN WHERE MATV = 1  UPDATE THANHVIEN  SET DIACHI = N’ 122, Nguyễn Văn Cừ, phường 10, quận 5, TPHCM’  COMMIT |

1. Chạy lỗi trên SQL:

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

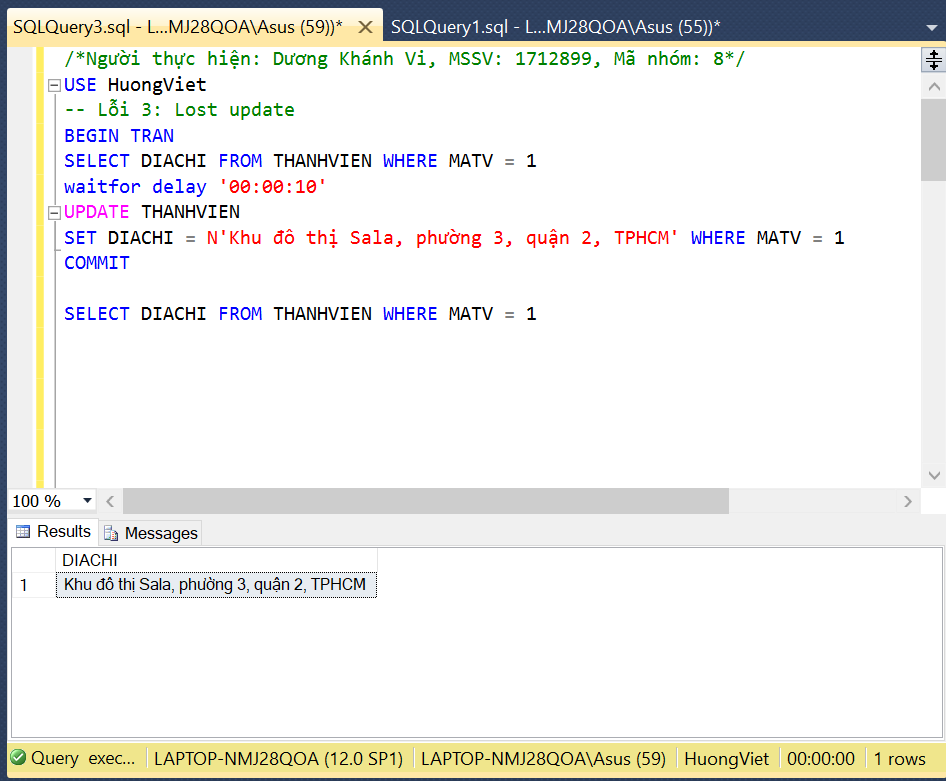


Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy lệnh Select của giao tác 2 và ta có kết quả:



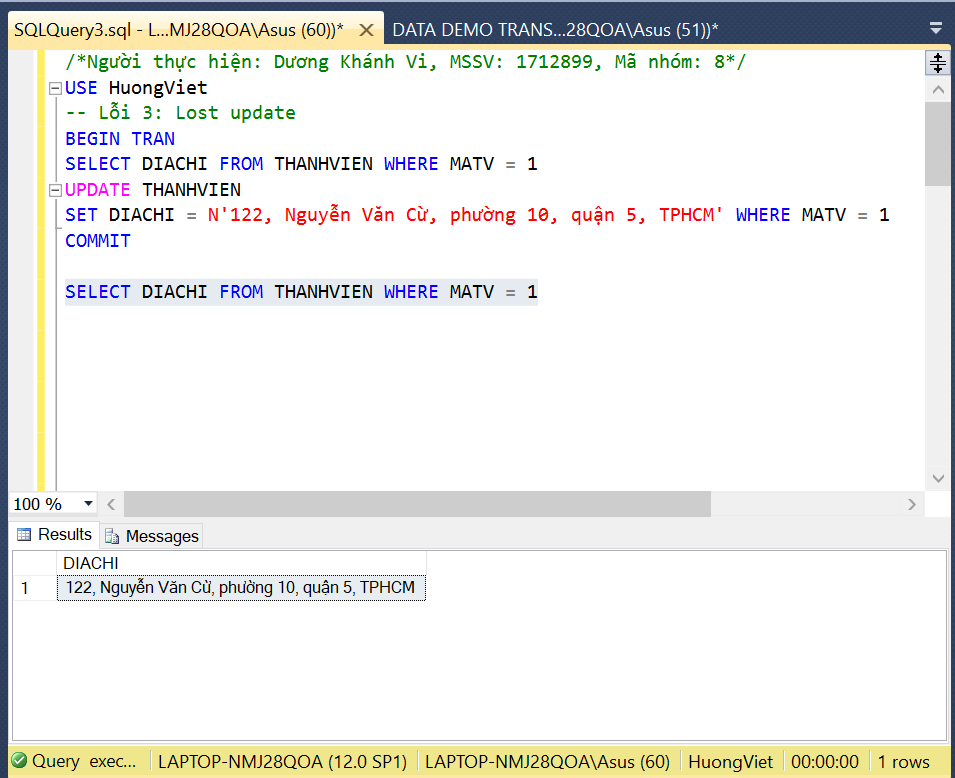
* 2 giao tác đều đọc được dữ liệu giống nhau, cũng là dữ liệu ban đầu.

Giao tác 1 cập nhật địa chỉ và commit:



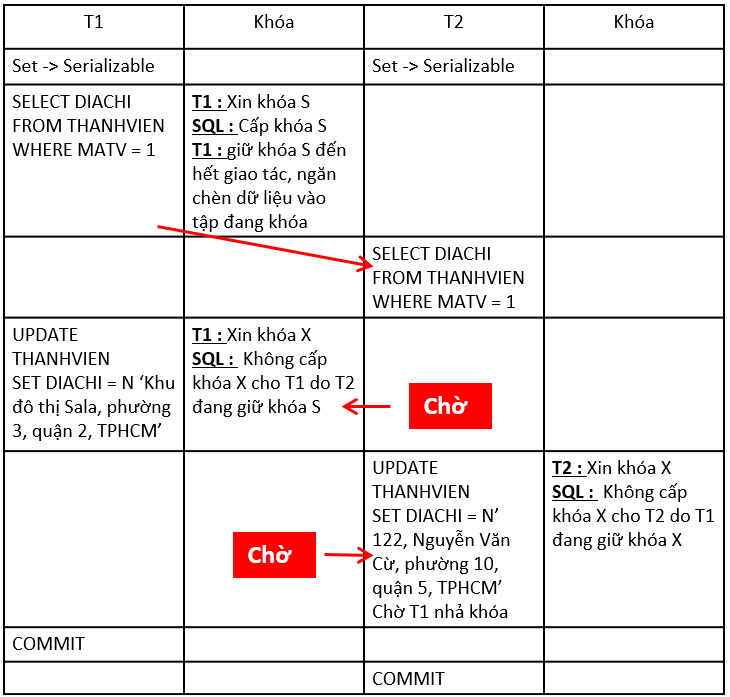
* Dữ liệu đã được cập nhật thành công

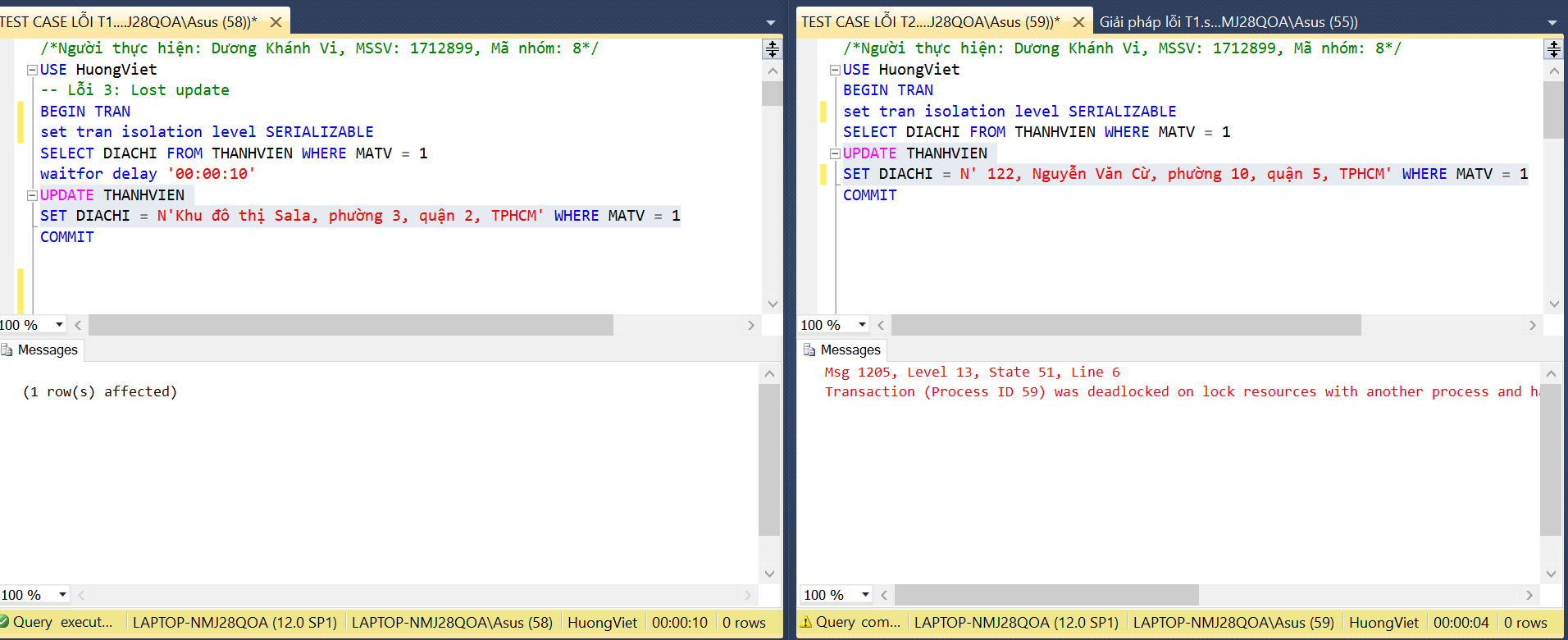
Giao tác 2 cập nhật địa chỉ và commit:



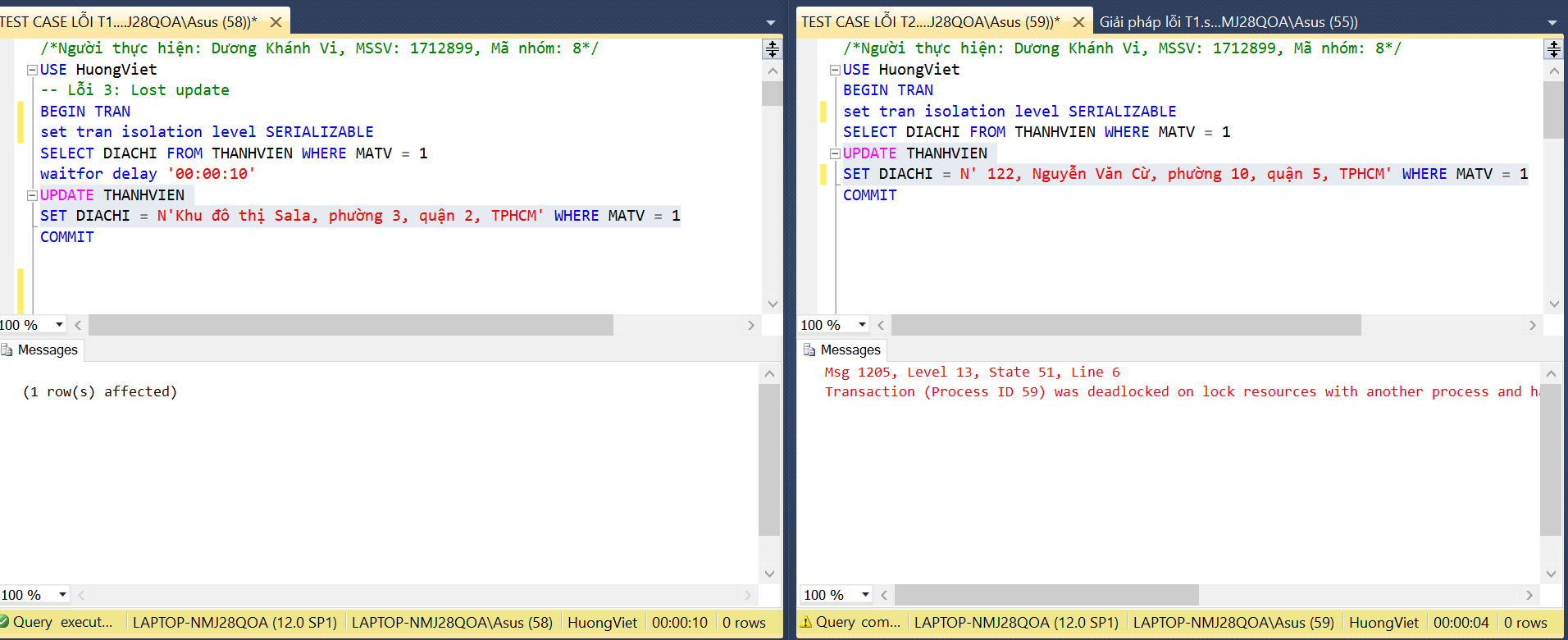
* Giao tác 2 đã cập nhật dữ liệu đè lên giao tác 1, nếu người dùng của giao tác 1 vào xem lại dữ liệu sẽ thấy có sự thay đổi.

1. ***Giải pháp:*** Set cơ chế khóa Serializable nhưng sẽ tạo thành Deadlock.

Đồ thị ưu tiên: 

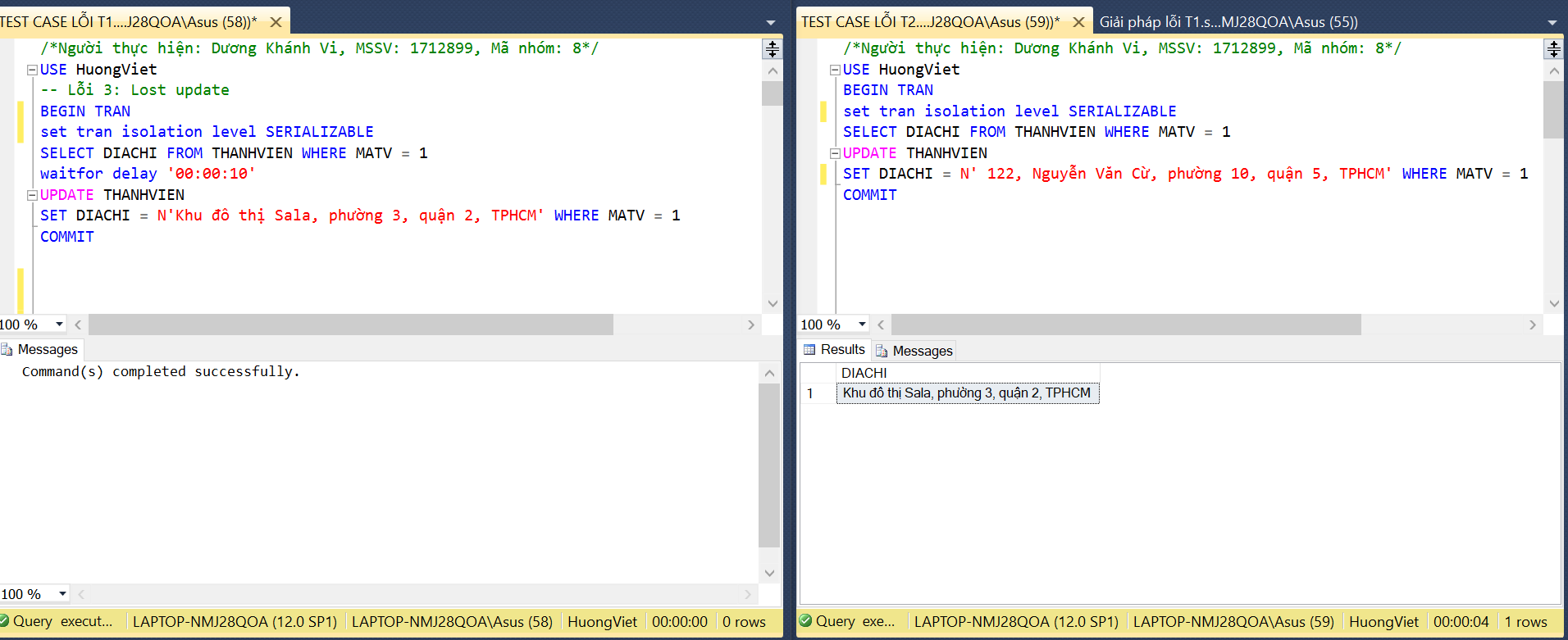


1. ***Deadlock:***
2. ***Lỗi:*** Lỗi Lost Update ở trên nhưng được giải quyết bằng cơ chế khóa Serializable, dẫn đến Deadlock



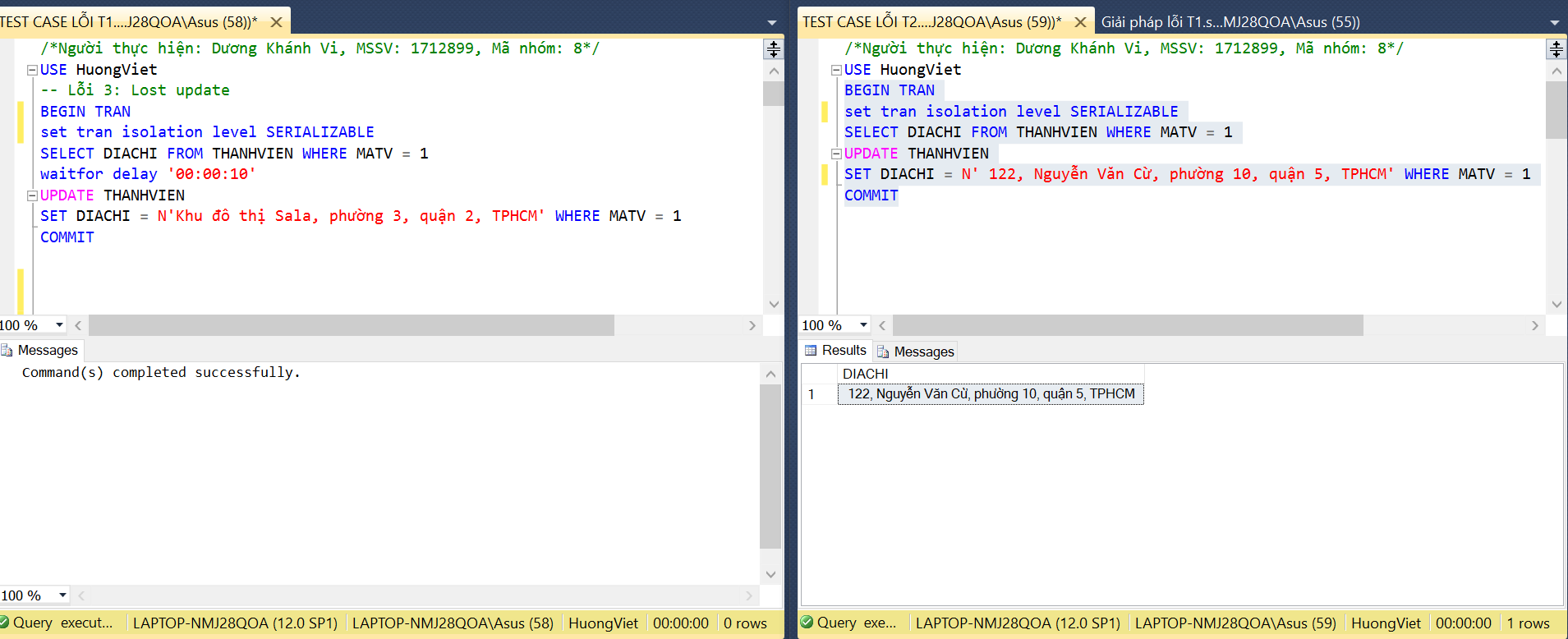
1. ***Giải pháp:*** Để giải quyết được Deadlock, Hệ quản trị CSDL đã tự động thực hiện 1 giao tác và hủy bỏ một giao tác còn lại. Cụ thể là thực hiện giao tác T1 và hủy bỏ T2 và yêu cầu người sử dụng chạy lại sau.

T1 commit giao tác:



* T1 sau khi commit, T2 không hiện lỗi deadlock nữa

Thực hiện lại T2:



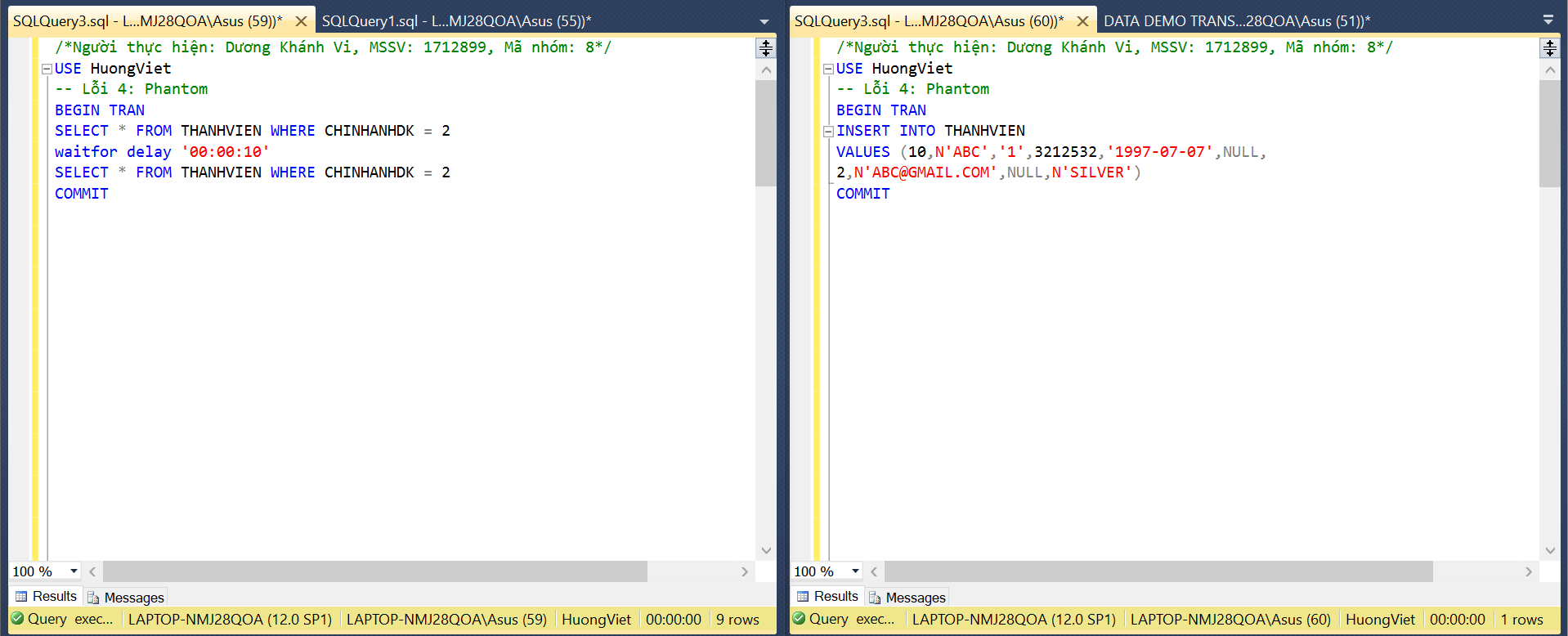
* T2 đã cập nhật dữ liệu thành công.

1. ***Phantom:***
2. ***Lỗi:*** Giả sử nhân viên quản lý khách hàng đang xem lại thông tin của tất cả khách hàng đã đăng ký thành viên trong chi nhánh 2. Trong lúc này, một khách hàng khác vào và đăng ký thành viên trong chi nhánh 2 bằng kênh online. Khi nhân viên xem lại thì thấy số lượng khách hàng đăng ký thành viên trong chi nhánh 2 bị thay đổi.

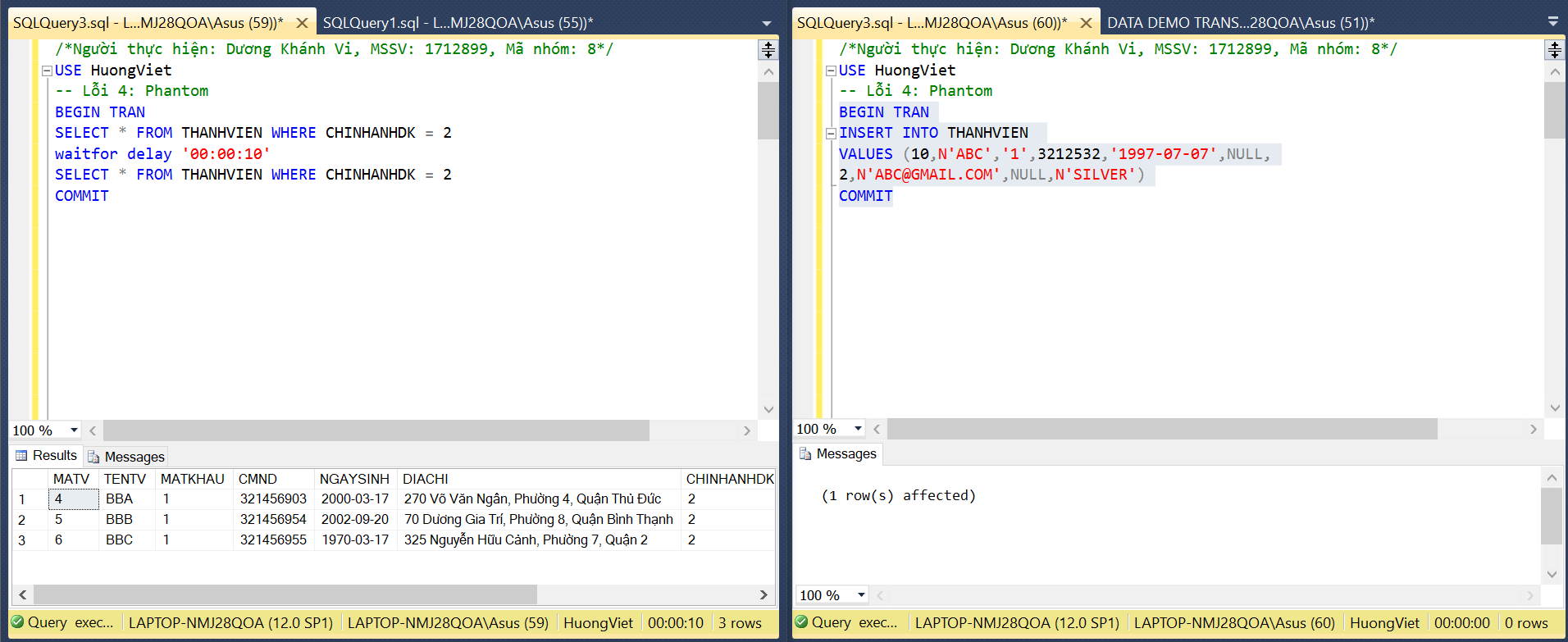
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  SELECT \* FROM THANHVIEN WHERE MACN = 2  waitfor delay‘00:00:10’  SELECT \* FROM THANHVIEN WHERE MACN = 2  COMMIT | BEGIN TRAN  INSERT INTO THANHVIEN VALUES (10,N'ABC','1','3212532','1997-07-07',NULL,  2,'ABC@GMAIL.COM',NULL,N'SILVER')  COMMIT |

1. Chạy lỗi trên SQL:

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

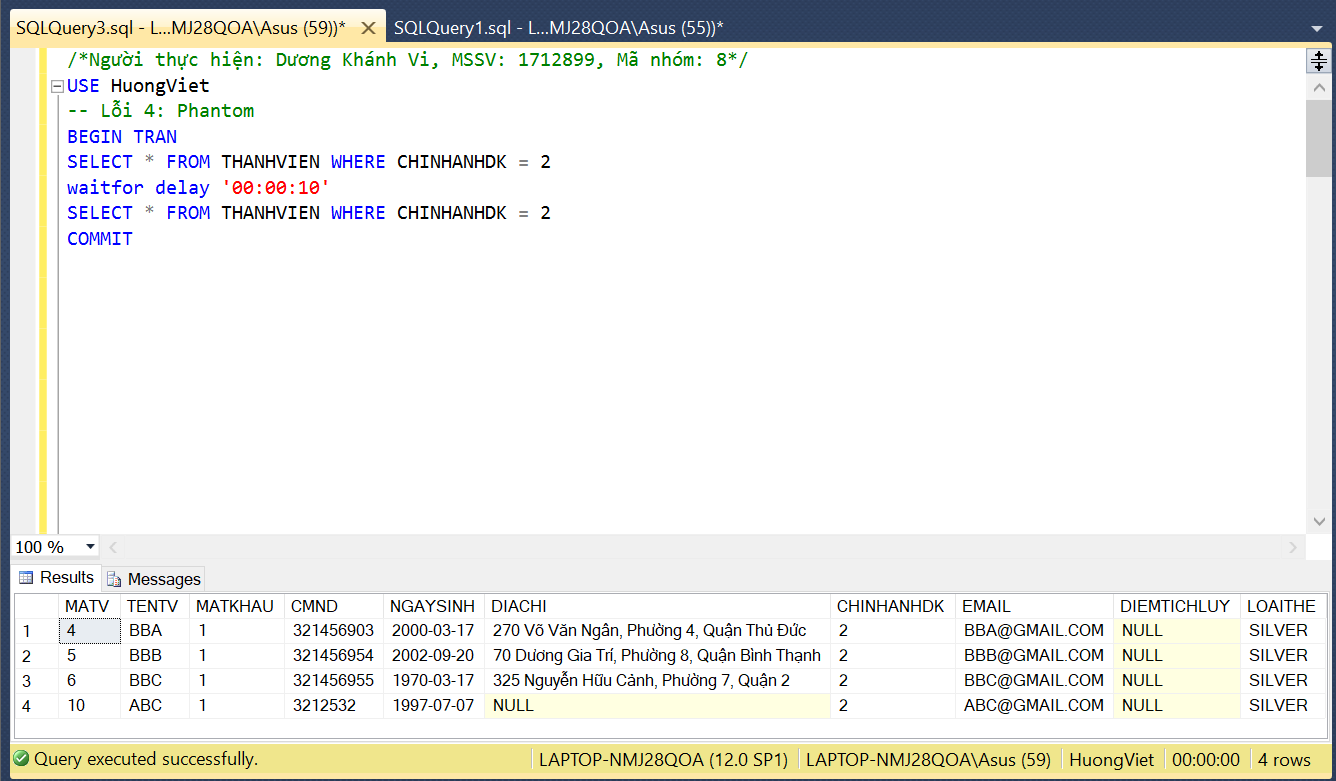


Chạy giao tác 1 đến lệnh waitfor delay ’00:00:10’, sau đó chạy hết tác 2 và ta có kết quả:



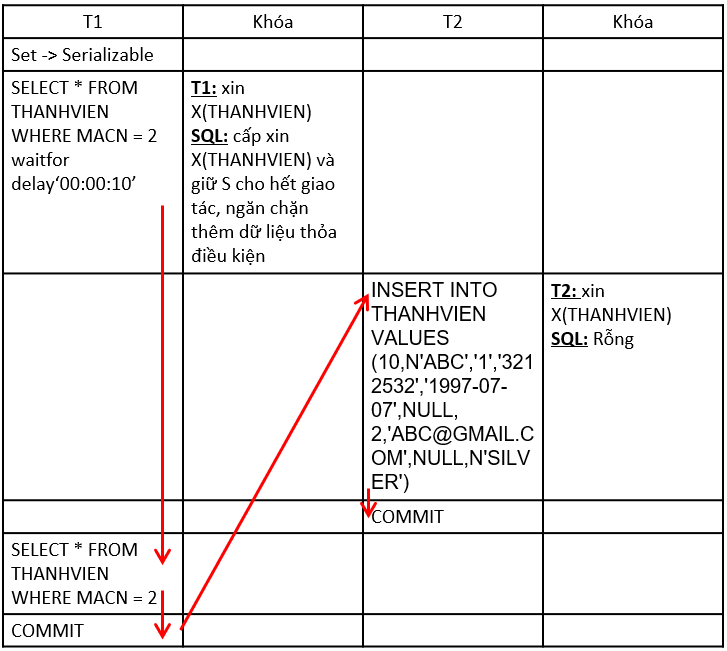
* Giao tác 1 xem dữ liệu của chi nhánh 2 và nhận được dữ liệu ban đầu (không cập nhật/ thêm). Giao tác 2 đã thêm dữ liệu thành công.

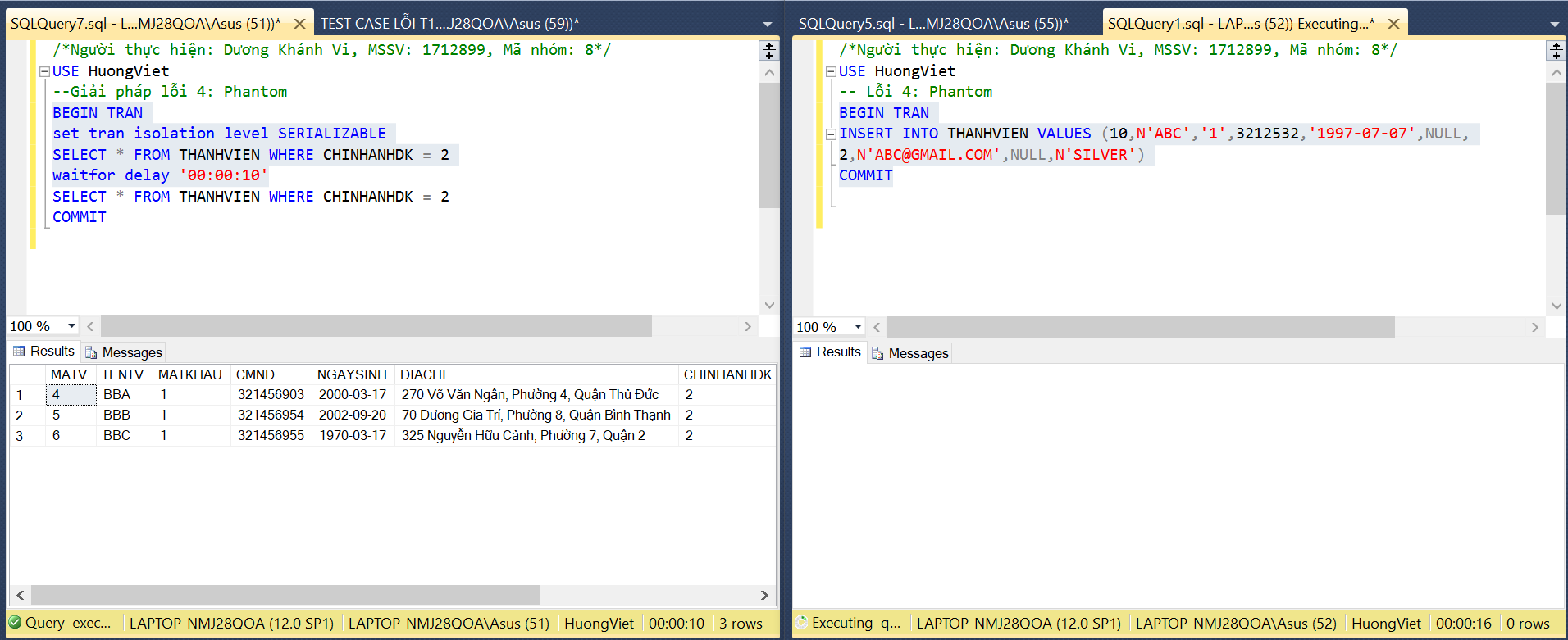
Giao tác 1 tiến hành xem dữ liệu của chi nhánh 2 một lần nữa:



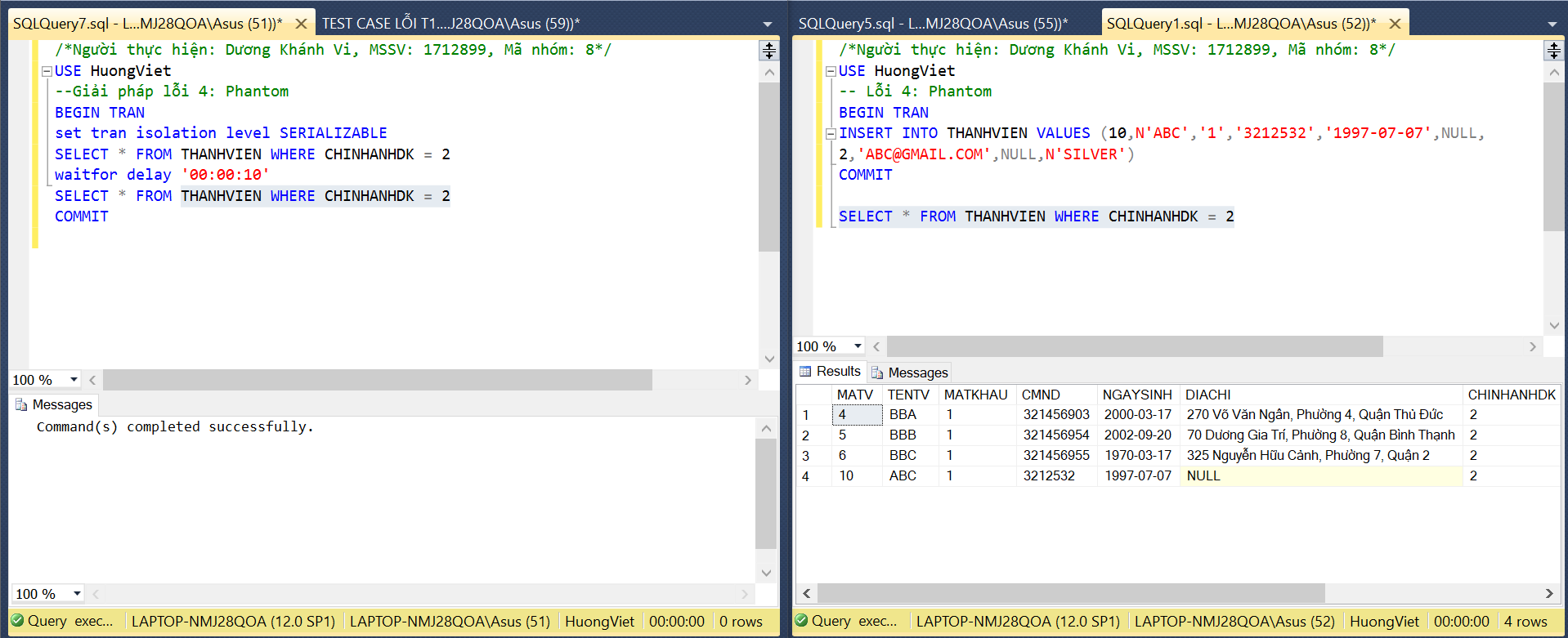
* Giao tác 1 đọc dữ liệu và thấy kết quả bị thêm 1 dòng.

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



Chạy giao tác 1 đến lệnh waitfor delay ’00:00:15’, 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