|  |  |  |  |
| --- | --- | --- | --- |
| Tên thủ tục | Mục đích | Input | Output |
| sp\_XemThucDon | Để xem thực đơn của một chi nhánh trong một ngày | @MACN,@Ngay | Gồm mã món, tên món, tên loại và số lượng |
| sp\_TiLeHuyDonTheoKenh | Tính tỉ lệ hủy đơn theo các kênh đặt hàng trong một chi nhánh | @MACN | Gồm 3 cột là tỉ lệ hủy Online, tỉ lệ hủy điện thoại và tỉ lệ hủy trực tiếp |
| sp\_TiLeHuyDonTheoLoaiKH | Tính tỉ lệ hủy đơn theo loại khách hàng trong một chi nhánh | @MACN | Gồm 2 cột là tỉ lệ hủy theo khách vãng lai và tỉ lệ hủy theo thành viên |
| sp\_TiLeHuyDonTheoKenhTong | Tỉ lệ hủy đơn theo kênh đặt hàng của toàn chi nhánh |  | Gồm Online, điện thoại và trực tiếp |
| sp\_TiLeHuyDonTheoLoaiKHTong | Tỉ lệ hủy đơn theo loại khách của toàn chi nhánh |  | Gồm khách vãng lai và thành viên |
| sp\_ThemMon | Thêm món vào bảng MONAN | @TENMON, @MALOAI, @GiA | Thêm dữ liệu vào bảng MONAN |
| sp\_XoaMon | Xóa món khỏi bảng MONANăn | @MAMON | Xóa dữ liệu khỏi bảng MONAN |
| sp\_SuaMon | Sửa món trong bảng MONAN | @MAMON, @TENMON, @MALOAI, @GiA | Sửa dữ liệu một món trong bảng MONAN |
| sp\_ThemLoaiMon | Thêm một loại món vào bảng LOAIMON | @TENLOAI | Thêm dữ liệu vảo bảng LOAIMON |
| sp\_XoaLoaiMon | Xóa một loại món khỏi bảng LOAIMON | @MALOAI | Xóa dữ liệu khỏi bảng LOAIMON |

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

|  |  |  |
| --- | --- | --- |
| Tên lỗi | Kịch bản lỗi | Xử lý |
| Dirty Read | Giả sử, do nguyên liệu để chế biến món có mã món 1 ở chi nhánh 1 không đủ và chưa kịp cung ứng. Nên quản lý chi nhánh 1 đã cập nhật số lượng còn lại của món 1 chỉ còn 10 phần nhưng chưa commit. Trong lúc này thì có một khách hàng online xem thực đơn ở chi nhánh 1 thì thấy với mã món số 1 thì số lượng còn lại là 10 nhưng cần đặt 11 phần nên không đặt nữa. Sau đó đã gặp sự cố nên giao tác của quản lý đã rollback không cập nhật thành công.  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 ở T2 |
| Unrepeatable Read | Khách hàng 1 thực hiện xem thực đơn của chi nhánh 1, thì thấy số lượng còn lại của món 1 là 50. Sau đó có khách hàng 2 tới đặt 5 phần món 1 ở chi nhánh 1 và được nhân viên lễ tân tiếp nhận đơn. Khách hàng 1 thực hiện xem thực đơn lại một lần nữa thì thấy số lượng còn lại của món 1 là 45. | Thiết lập mức cô lập Repeatable Read ở T1 |
| Lost Update | Giả sử quản lý chi nhánh 1 vào xem thực đơn chi nhánh 1 sau đó cập nhật số phần còn lại của món 1 là 45. Cùng lúc đó quản lý công ty vào xem thực đơn chi nhánh 1, sau đó cập nhật số phần còn lại của món 1 là 40.  Nếu sau đó quản lý chi nhánh vào xem lại thực đơn thì sẽ không hiện thông tin như mình vừa cập nhật. | Thiết lập mức cô lập Repeatable Read ở T1 và T2. Xảy ra deadlock và SQL sẽ ngừng T2 và bắt thực hiện lại sau |
| Phantom | Giả sử khách hàng xem thực đơn của chi nhánh 1 thấy chỉ có món 1. Sau đó, quản lý công ty có thêm món 2 vào thực đơn. Khi khách hàng xem lại thực đơn thì thấy xuất hiện thêm món 2. | Thiết lập mức cô lập Serializable ở T1 |
| Deadlock | Giả sử có 2 người quản lý công ty. Người thứ nhất cập nhật số phần ăn của món 1 ở chi nhánh 1 sau đó tiến hành xem thực đơn chi nhánh 2 cần chỉnh sửa gì không. Cùng lúc đó người thứ 2 cập nhật số phần ăn của món 1 ở chi nhánh 2 và sau đó tién hành xem thực đơn chi nhánh 1. Lúc này giao tác của người thứ 2 bị rollback và thực hiện lại theo cơ chế xử lí Deadlock của SQL | SQL chọn T2 là victim Deadlock và bắt thực hiện lại sau |

* ***Chi tiết thủ tục tranh chấp:***

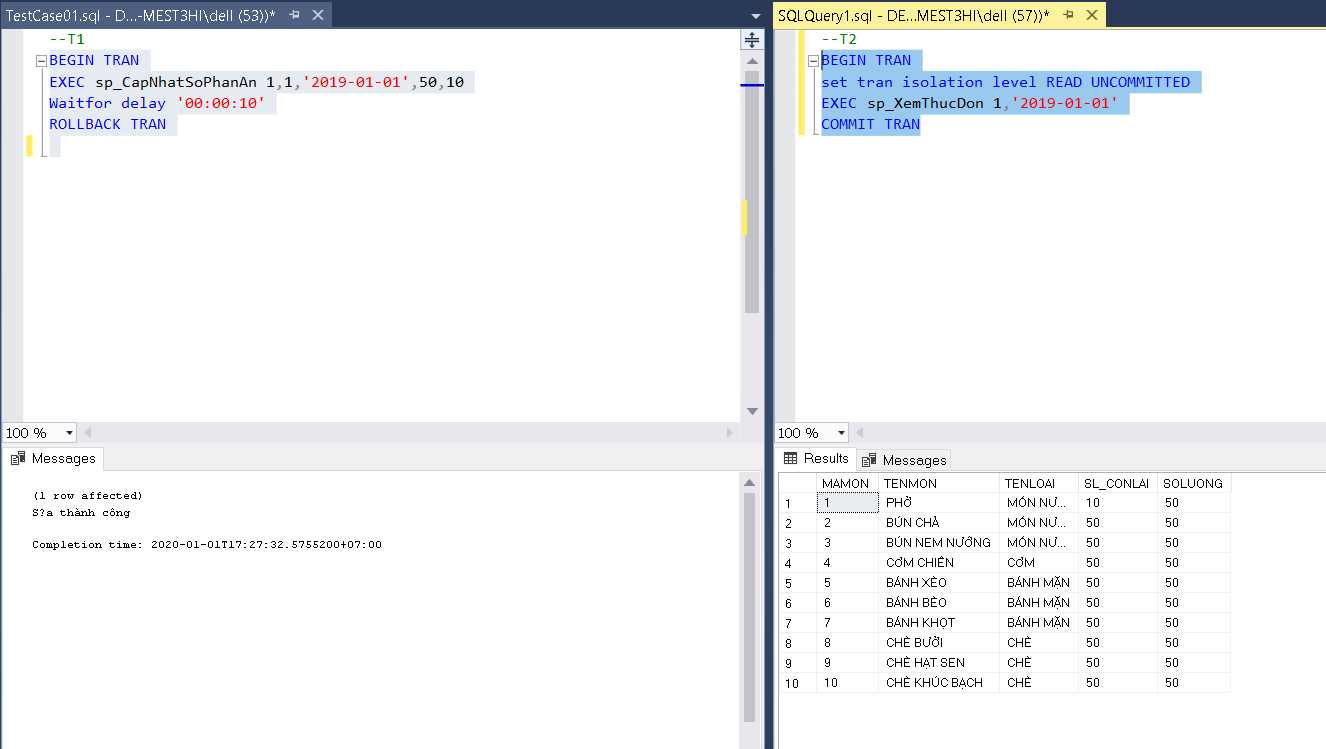
1. ***Dirty read:***
2. ***Lỗi :*** Giả sử, do nguyên liệu để chế biến món có mã món 1 ở chi nhánh 1 không đủ và chưa kịp cung ứng. Nên quản lý chi nhánh 1 đã cập nhật số lượng còn lại của món 1 chỉ còn 10 phần nhưng chưa commit. Trong lúc này thì có một khách hàng online xem thực đơn ở chi nhánh 1 thì thấy với mã món số 1 thì số lượng còn lại là 10 nhưng cần đặt 11 phần nên không đặt nữa. Sau đó đã gặp sự cố nên giao tác của quản lý đã rollback không cập nhật thành công. 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\_CapNhatSoPhanAn 1,1,'2019-01-01',50,10  Waitfor delay '00:00:10'  ROLLBACK TRAN | BEGIN TRAN  set tran isolation level READ UNCOMMITTED  EXEC sp\_XemThucDon 1,'2019-01-01'  COMMIT TRAN |

1. Chạy lỗi trên SQL:

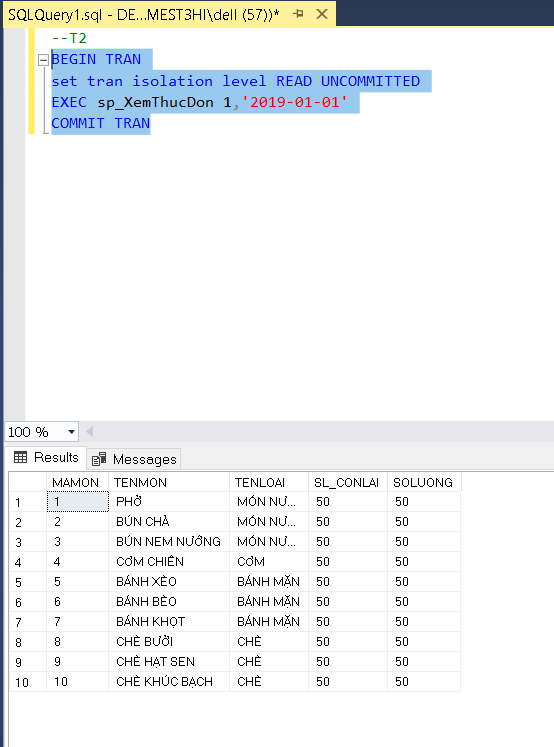
**Đoạn test case của trường hợp Dirty Read:**

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 Số lượng còn lại của món ăn có mã 1 là 10.

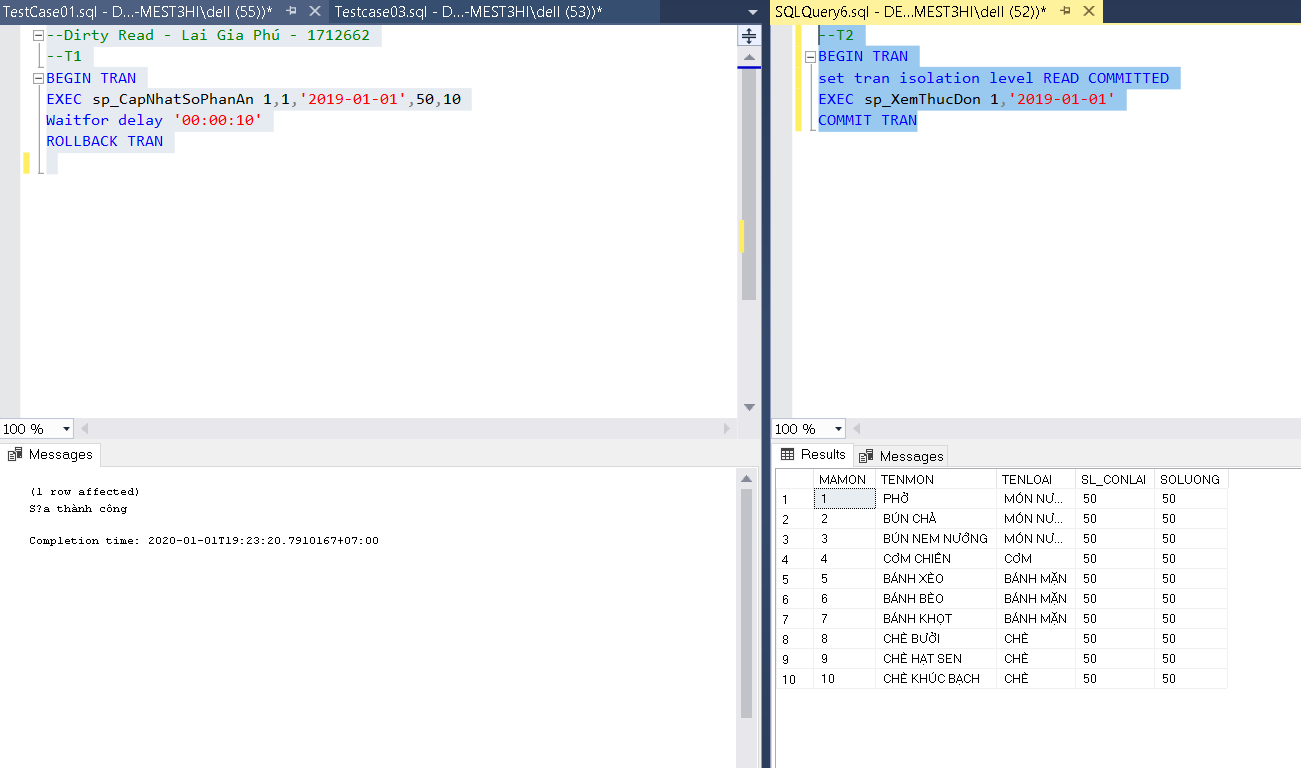
Sau khi Giao tác 1 rollback, thử truy vấn lại thì số lượng còn lại đã trở lại 50.



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 giữ khóa X và chưa kết thúc giao tác.

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



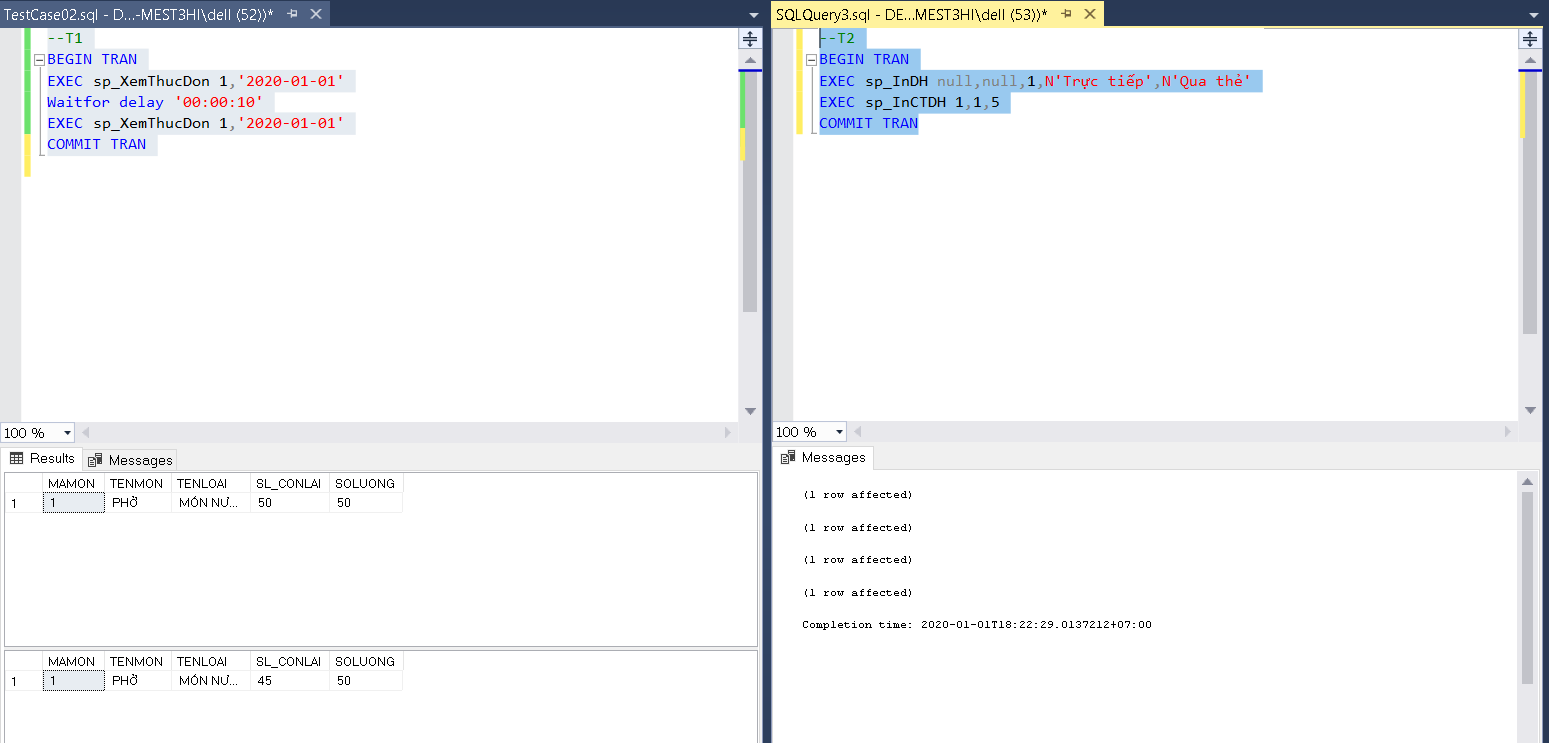
1. ***Unrepeatable read:***
2. ***Lỗi:*** Khách hàng 1 thực hiện xem thực đơn của chi nhánh 1, thì thấy số lượng còn lại của món 1 là 50. Sau đó có khách hàng 2 tới đặt 5 phần món 1 ở chi nhánh 1 và được nhân viên lễ tân tiếp nhận đơn. Khách hàng 1 thực hiện xem thực đơn lại một lần nữa thì thấy số lượng còn lại của món 1 là 45.

|  |  |
| --- | --- |
| 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\_InDH null,null,1,N'Trực tiếp',N'Qua thẻ'  EXEC sp\_InCTDH 1,1,5  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:



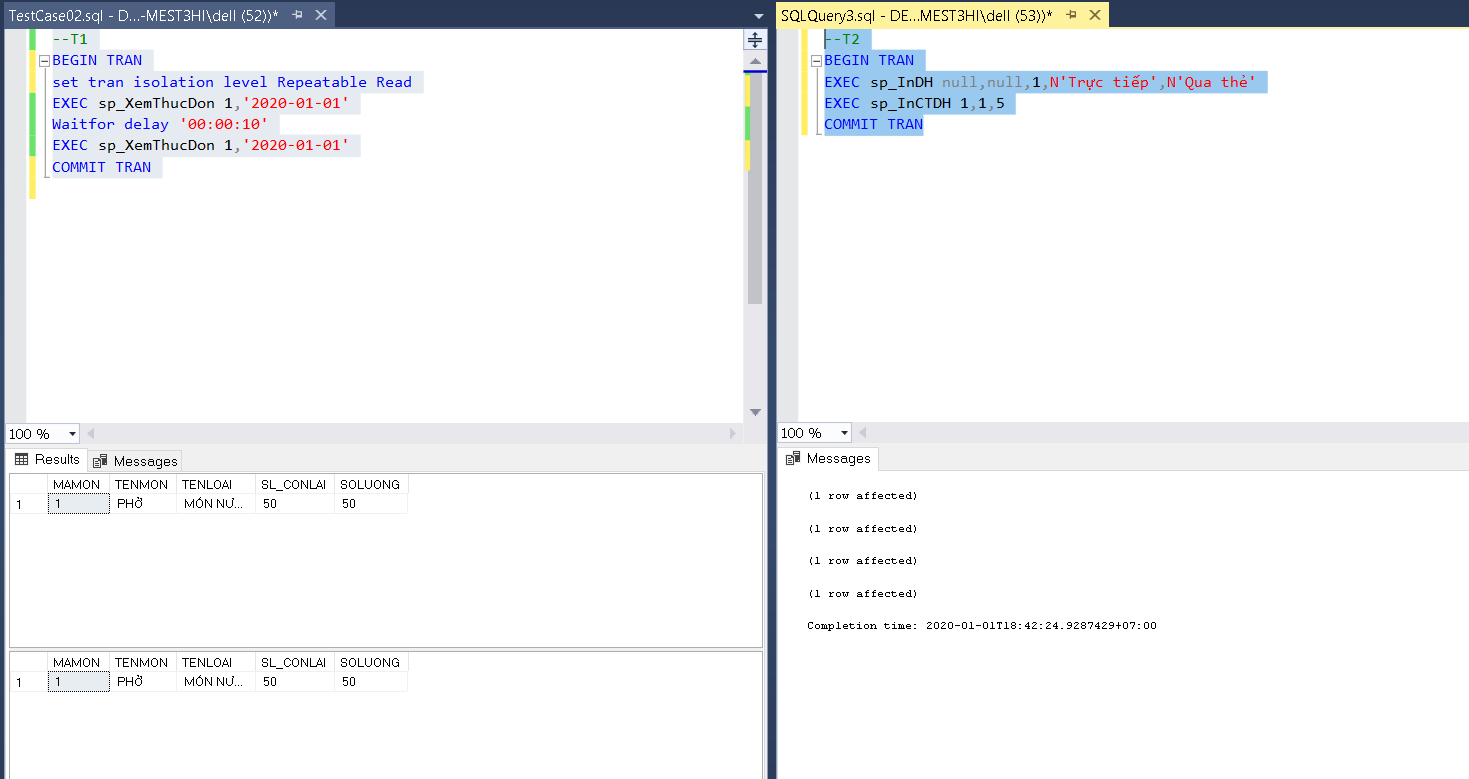
* Giao tác 1 tiến hành xem thực đơn chi nhánh 1 kết quả giống hệt dữ liệu ban đầu(chưa chỉnh sửa).

Giao tác 1 xem thực đơn chi nhánh 1 lần 2:

* Ở lần đọc thứ 2 này, Giao tác 1 đã thấy dữ liệu có sự thay đổi, cụ thể là số lượng còn lại chỉ còn 45

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 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. Khi T1 vừa hoàn thành thì T2 mới cập nhật số lượng còn lại.

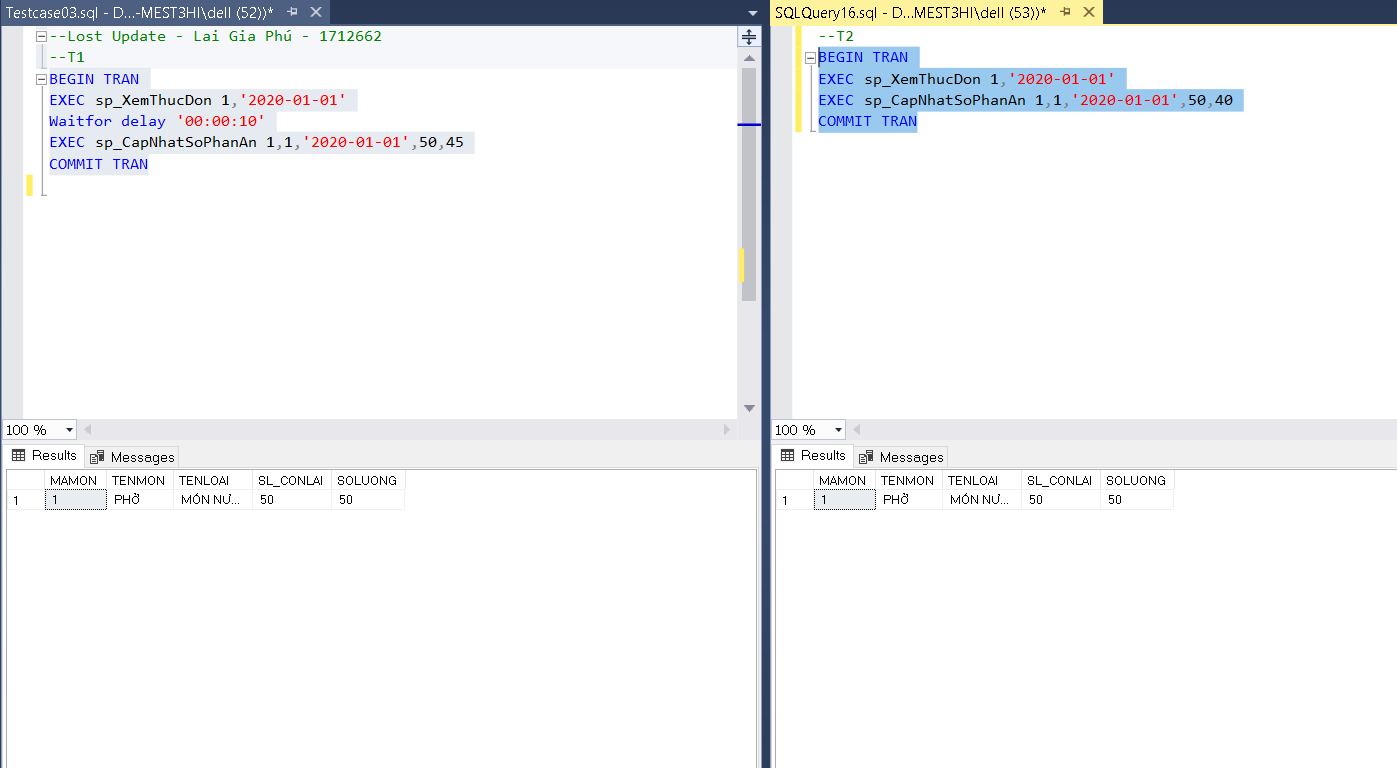


1. ***Lost update:***
2. ***Lỗi:*** Giả sử quản lý chi nhánh 1 vào xem thực đơn chi nhánh 1 sau đó cập nhật số phần còn lại của món 1 là 45. Cùng lúc đó quản lý công ty vào xem thực đơn chi nhánh 1, sau đó cập nhật số phần còn lại của món 1 là 40.Nếu sau đó quản lý chi nhánh vào xem lại thực đơn thì sẽ không hiện thông tin như mình vừa cập nhật.

|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_XemThucDon 1,'2020-01-01'  Waitfor delay '00:00:10'  EXEC sp\_CapNhatSoPhanAn 1,1,'2020-01-01',50,45  COMMIT TRAN | BEGIN TRAN  EXEC sp\_XemThucDon 1,'2020-01-01'  EXEC sp\_CapNhatSoPhanAn 1,1,'2020-01-01',50,40  COMMIT TRAN |

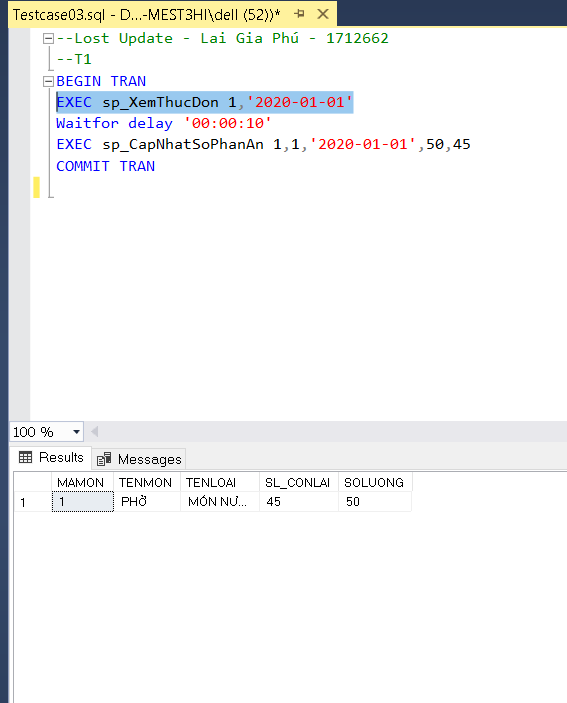
1. Chạy lỗi trên SQL:

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



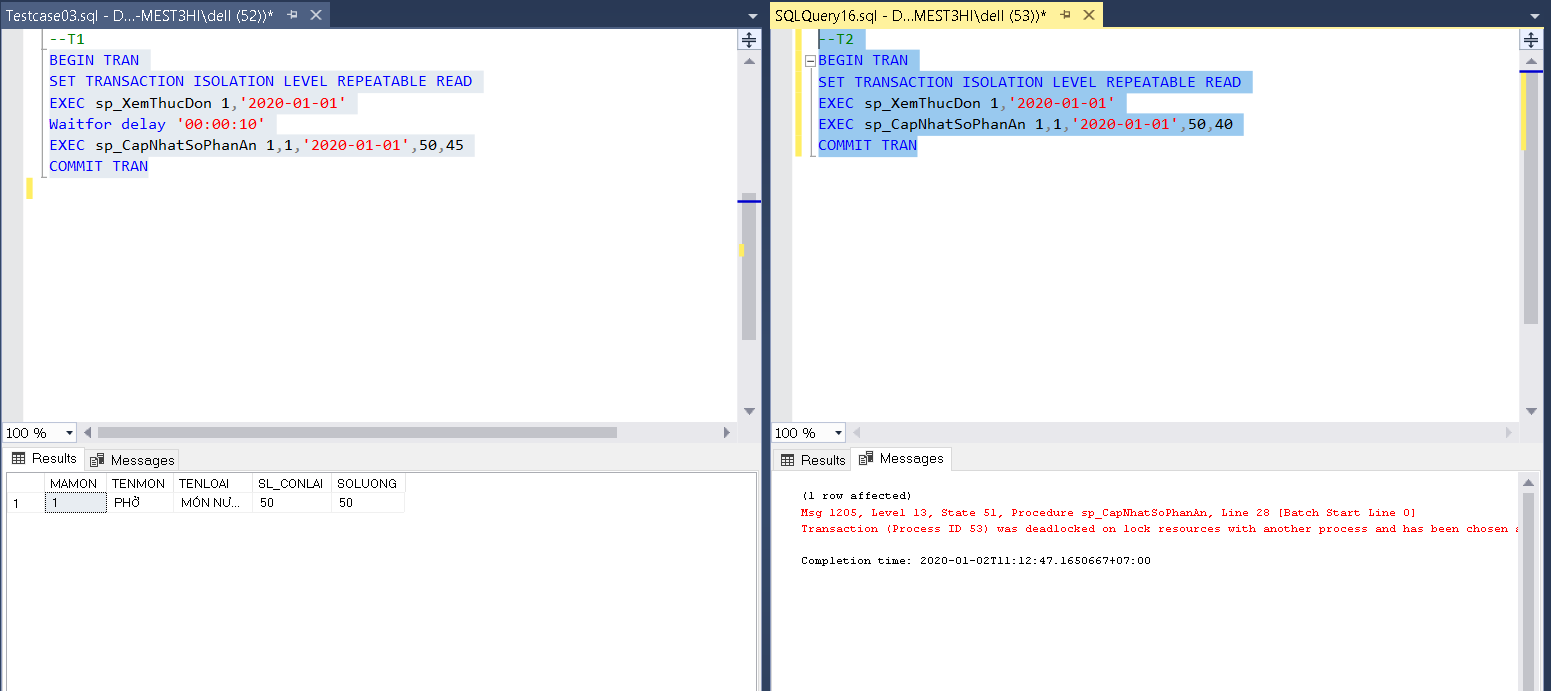
* 2 giao tác đều đọc được dữ liệu giống nhau, cũng là dữ liệu ban đầu.

Dữ liệu thật sự sau cập nhật



* Giao tác 1 đã cập nhật dữ liệu đè lên giao tác 2, nếu người dùng của giao tác 2 vào xem lại dữ liệu sẽ không giống như dữ liệu họ cập nhật.

1. ***Giải pháp:*** Thiết lập mức cô lập Repeatable Read ở T1 và T2. Xảy ra deadlock và SQL sẽ ngừng T2 và bắt thực hiện lại sau



Đồ thị ưu tiên: T1 -> T2

1. ***Phantom:***
2. ***Lỗi:*** Giả sử khách hàng xem thực đơn của chi nhánh 1 thấy chỉ có món 1. Sau đó, quản lý công ty có thêm món 2 vào thực đơn. Khi khách hàng xem lại thực đơn thì thấy xuất hiện thêm món 2.

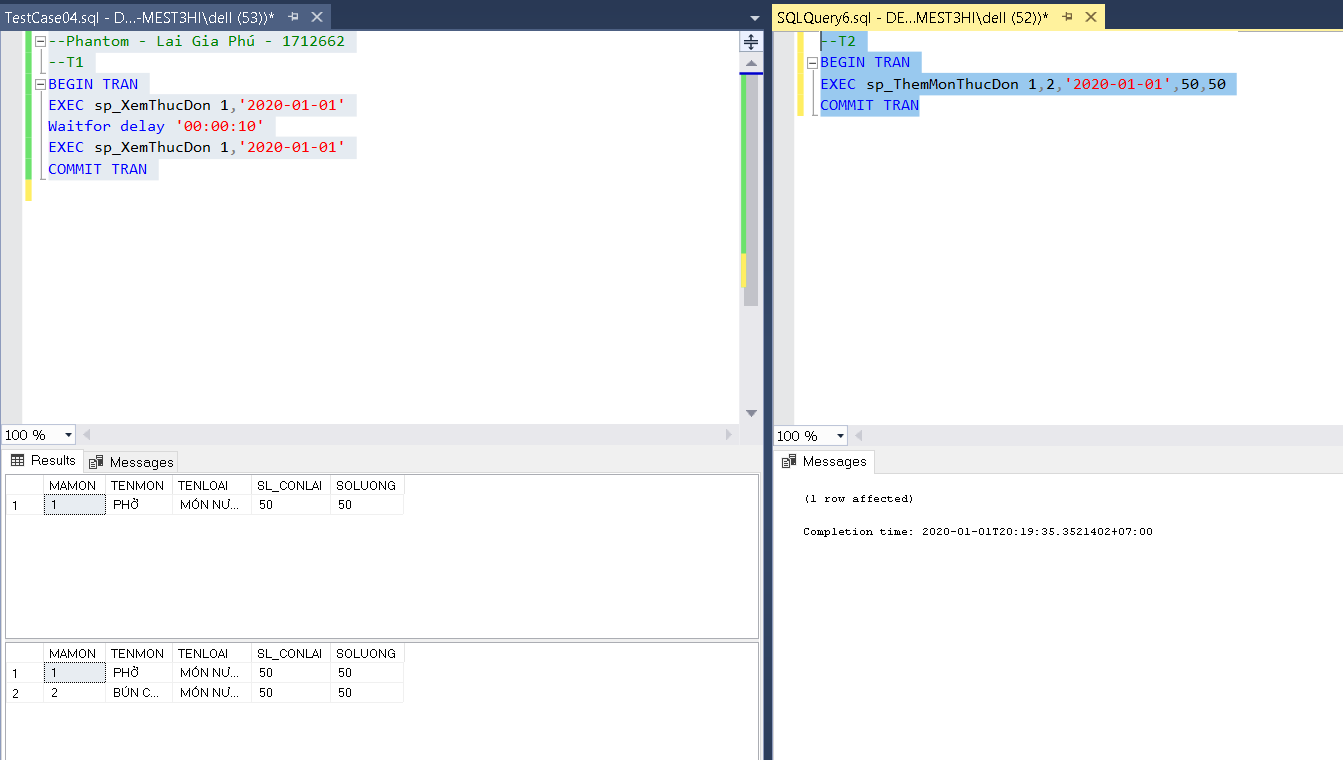
|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_XemThucDon 1,'2020-01-01'  Waitfor delay '00:00:10'  EXEC sp\_XemThucDon 1,'2020-01-01'  COMMIT TRAN | BEGIN TRAN  EXEC sp\_ThemMonThucDon 1,2,'2020-01-01',50,50  COMMIT TRAN |

1. Chạy lỗi trên SQL:

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

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

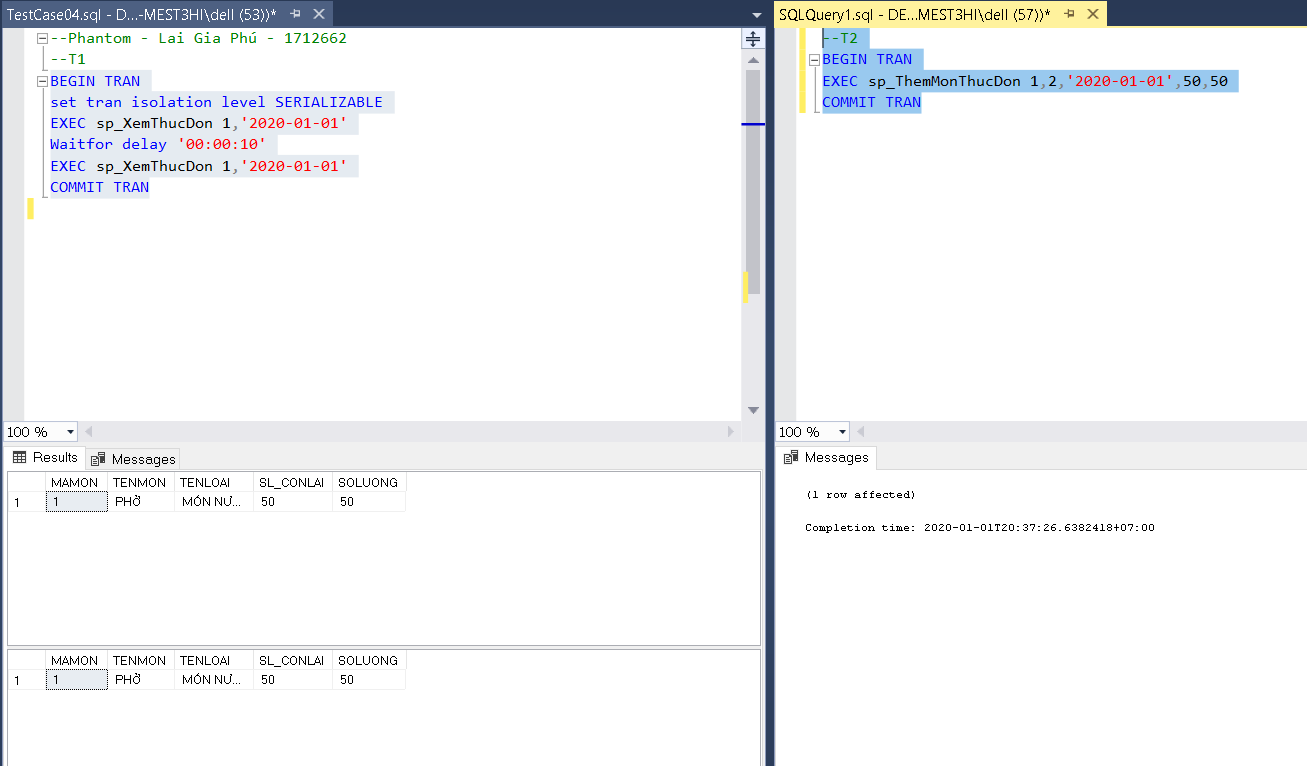
Giao tác 1 tiến hành xem thực đơn của chi nhánh 1 một lần nữa:



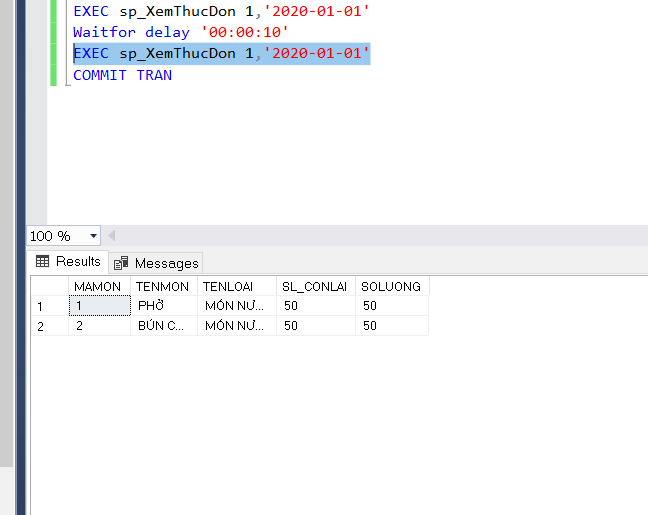
* Giao tác 1 đọc dữ liệu và thấy thêm 1 món

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.



Sau khi T2 commit

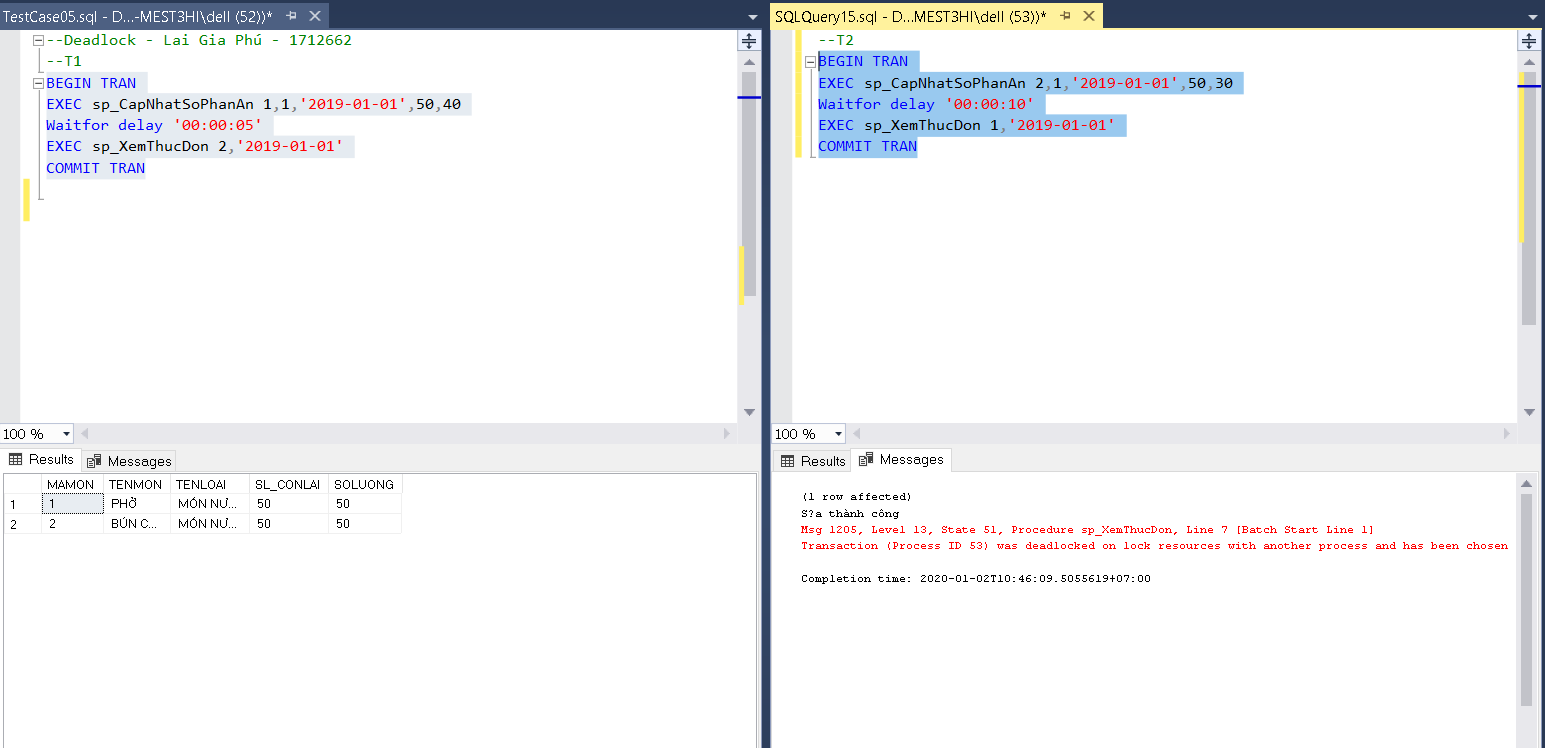


1. ***Deadlock:***
2. ***Lỗi:*** Giả sử có 2 người quản lý công ty. Người thứ nhất cập nhật số phần ăn của món 1 ở chi nhánh 1 sau đó tiến hành xem thực đơn chi nhánh 2 cần chỉnh sửa gì không. Cùng lúc đó người thứ 2 cập nhật số phần ăn của món 1 ở chi nhánh 2 và sau đó tién hành xem thực đơn chi nhánh 1. Lúc này giao tác của người thứ 2 bị rollback và thực hiện lại theo cơ chế xử lí Deadlock của SQL

|  |  |
| --- | --- |
| T1 | T2 |
| BEGIN TRAN  EXEC sp\_CapNhatSoPhanAn 1,1,'2019-01-01',50,40  Waitfor delay '00:00:05'  EXEC sp\_XemThucDon 2,'2019-01-01'  COMMIT TRAN | BEGIN TRAN  EXEC sp\_CapNhatSoPhanAn 2,1,'2019-01-01',50,30  Waitfor delay '00:00:10'  EXEC sp\_XemThucDon 1,'2019-01-01'  COMMIT TRAN |

1. Chạy lỗi trên SQL:

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



Đồ thị ưu tiên: T1 -> T2

1. ***Giải pháp:*** SQL chọn T2 là victim Deadlock và bắt thực hiện lại sau

