**Mục lục**

[**1.** **Tổng quan dự án** 2](#_Toc92293520)

[**2.** **Kịch bản mua hàng – chứng minh tính đồng bộ** 2](#_Toc92293521)

[**3.** **Kịch bản quản lý – chứng minh phân mảnh và đồng bộ** 3](#_Toc92293522)

[**4.** **Kịch bản chứng minh tính trong suốt** 4](#_Toc92293523)

[**5.** **Kịch bản phân quyền trong hệ thống** 5](#_Toc92293524)

[**6.** **Kịch bản xác minh thông tin khác chi nhánh** 5](#_Toc92293525)

[**7.** **Kịch bản Concurrency & Commit Protocols** 5](#_Toc92293526)

[***1.1.*** ***Two transactions entered in a non-serializable order*** 5](#_Toc92293527)

[***1.2.*** ***Two transactions affect on the same tables -different rows*** 10](#_Toc92293528)

[***1.3.*** ***Read/write conflict*** 14](#_Toc92293529)

[***1.4.*** ***Write/write conflict*** 17](#_Toc92293530)

[**8.** **Kịch bản Distributed Failure/Recovery** 21](#_Toc92293531)

[***1.1.*** ***Correct recovery after a commit*** 21](#_Toc92293532)

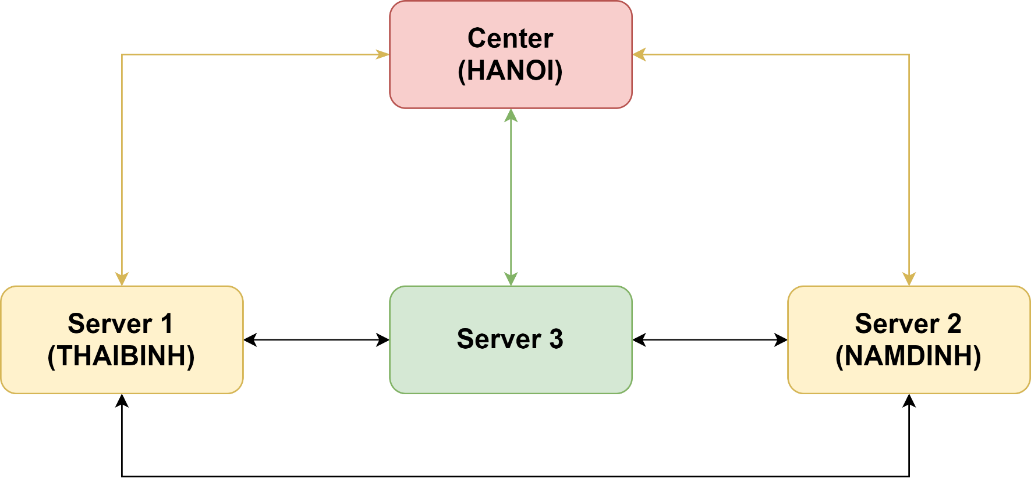
[***1.2.*** ***Correct recovery before a commit*** 25](#_Toc92293533)

[***1.3.*** ***A server site is crashed*** 27](#_Toc92293534)

**Kịch bản**

# **Tổng quan dự án**

* **Tên dự án:** Xây dựng hệ thống CSDL phân tán để quản lý công ty hải sản
* **Chi tiết:**
  + Gồm 1 trụ sở chính – tại Hà Nội – cũng là nơi đặt máy chủ HNSERVER
  + 2 chi nhánh tại: Thái Bình, Nam Định (mỗi chi nhánh địa phương có thể có nhiều kho)
  + Bao gồm 3 máy trạm, trong đó có 2 máy trạm TB1SERVER và ND2SERVER đặt tại Thái Bình và Nam Định. Và 1 máy trạm HN2SERVER đặt tại Hà Nội
  + *Vừa nói vừa show thầy xem sơ đồ này – lúc vào demo là bật ảnh sơ đồ này lên luôn rồi nói nhanh thôi:*



* + *Sau đó Show thầy xem máy chủ, 3 máy trạm và các bảng trên từng máy*

# **Kịch bản mua hàng – chứng minh tính đồng bộ**

* Khách hàng **A** tại **Nam Định** muốn mua “Cá Nục” và “Mực 1 Nắng” (mua xỉ - nhà sản xuất, nhà cung cấp thì không bán lẻ),   
  nhân viên chi nhánh kiểm tra giá và tình trạng các mặt hàng trên trong kho:

|  |
| --- |
| **SELECT NAME\_SEA, PRICE, INSTOCK FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA IN (N'Cá Nục', N'Mực 1 Nắng')** |

* Khách hàng muốn tham khảo thêm sản phẩm “Cá Thu Khô”, nhân viên quản lý chi nhánh liền kiểm tra:

|  |
| --- |
| **SELECT NAME\_SEA, PRICE, INSTOCK FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA = N'Cá Thu Khô'** |

* Khách hàng quyết định mua **5 cân mực 1 nắng** và **20 cân cá nục**. Nhân viên xin số điện thoại khách hàng để kiểm tra xem khách hàng đã tồn tại chưa (Tôi đi nhập hàng họ cũng hỏi số điện thoại để kiểm tra xem khách hàng thế nào),

Khách hàng đọc số điện thoại: 0988585568, Nhân viên kiểm tra:

|  |
| --- |
| **SELECT ID\_CUS FROM HN2SERVER.CHHS.dbo.CUSTOMER WHERE NUM = N'0988585568'** |

Nếu khách hàng đã tồn tại, Nhân viên nhấn vào ID khách hàng và nhập thông tin các mặt hàng và số lượng mỗi mặt hàng để hệ thống tự động tính tổng giá tiền (xử lý trên code) và báo cho khách hàng:

|  |
| --- |
| **SELECT PRICE \* 5 FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA = N'Mực 1 Nắng'**  **SELECT PRICE \* 20 FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA = N'Cá Nục'** |

\*Họ tên, địa chỉ, sđt khách hàng hệ thống đã tự nhập từ dữ liệu có sẵn theo ID. Còn nếu khách hàng chưa có sẵn thì nhân viên phải nhập thủ công trên màn hình hóa đơn.

Sau khi khách hàng đồng ý, nhân viên chọn xác nhận hóa đơn. Hệ thống thực hiện lần lượt:

|  |
| --- |
| **INSERT INTO CHHS.dbo.INVOICE (ID\_INV, TOTALPRICE , ID\_CUS, ID\_EMP, ID\_STO)**  **VALUES (1026, 2350000, 'CUS06', 'EMP5', 'KD3')**  **GO**  **INSERT INTO CHHS.dbo.INVOICE\_DETAIL(ID\_INV, ID\_SEA, AMOUNT)**  **VALUES (1026, (SELECT ID\_SEA FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA = N'Mực 1 Nắng'), 5)**  **GO**  **INSERT INTO CHHS.dbo.INVOICE\_DETAIL(ID\_INV, ID\_SEA, AMOUNT)**  **VALUES (1026, (SELECT ID\_SEA FROM CHHS.dbo.SEAFOOD WHERE NAME\_SEA = N'Cá Nục'), 20)**  **GO**  **UPDATE CHHS.dbo.SEAFOOD SET INSTOCK -= 5 WHERE NAME\_SEA = N'Mực 1 Nắng'**  **GO**  **UPDATE CHHS.dbo.SEAFOOD SET INSTOCK -= 20 WHERE NAME\_SEA = N'Cá Nục'** |

* + *Sau đó Show thầy xem các máy trạm và máy chủ đã đồng bộ chưa*

# **Kịch bản quản lý – chứng minh phân mảnh và đồng bộ**

* Từ máy chủ, quản lý muốn xem thông tin chi tiết các nhân viên ở các chi nhánh Nam Định, sắp xếp theo Lương tăng dần (rồi theo tên giảm dần):

|  |
| --- |
| **SELECT \* FROM CHHS.dbo.EMPLOYEE**  **WHERE ID\_BRA = N'BRA2'**  **ORDER BY SAL ASC, FULLNAME DESC** |

* Do có khách thuộc khu vực gần kho Nam Định **đặt** **20 cân Trứng cá tầm** nhưng trong kho chỉ còn 12 cân, vì vậy quản lý quyết định **nhập thêm 30 cân Trứng cá tầm** từ công ty **5fine.co, USA** và **lưu trữ vào kho Nam Định 2** và nhân viên **Lê Ngọc Ánh sẽ chịu trách nhiệm** quá trình tiếp nhận. Hệ thống thực hiện:

|  |
| --- |
| **INSERT INTO CHHS.dbo.RECEIPT\_NOTE (ID\_REC, SOURCE, TOTALPRICE, ID\_EMP, ID\_STO)**  **VALUES (211, N'5fine.co, USA', 6000000, (SELECT ID\_EMP FROM CHHS.dbo.EMPLOYEE WHERE FULLNAME = N'Lê Ngọc Ánh'), (SELECT ID\_STO FROM CHHS.dbo.STORAGE WHERE NAME\_STO = N'Nam Định 2'))**  **GO**  **IF (SELECT ID\_SEA FROM CHHS.dbo.SEAFOOD WHERE (NAME\_SEA = N'Trứng Cá Tầm' AND SUPPLIER = N'5fine.co, USA')) != N''**  **UPDATE CHHS.dbo.SEAFOOD SET INSTOCK += 30 WHERE (NAME\_SEA = N'Trứng Cá Tầm' AND SUPPLIER = N'5fine.co, USA');**  **ELSE**  **INSERT INTO CHHS.dbo.SEAFOOD (ID\_SEA, NAME\_SEA, PRICE, SUPPLIER, INSTOCK)**  **VALUES('VI5', N'Trứng Cá Tầm', 200000, N'5fine.co, USA', 30);**  **GO**  **INSERT INTO CHHS.dbo.RECEIPT\_DETAIL (ID\_REC, ID\_SEA, AMOUNT, PRICE)**  **VALUES (211, (SELECT ID\_SEA FROM CHHS.dbo.SEAFOOD WHERE (NAME\_SEA = N'Trứng Cá' AND SUPPLIER = N'5fine.co, USA')), 30, 200000);** |

* Quản lý muốn thống kê tổng số lượng khách hàng chi nhánh Thái Bình bán được

|  |
| --- |
| **SELECT COUNT(\*) FROM CHHS.dbo.CUSTOMER WHERE ID\_BRA = N'BRA1'** |

* Quản lý muốn xem Các nhân viên và tên Các chi nhánh tương ứng

|  |
| --- |
| **SELECT EMPLOYEE.FULLNAME, BRANCH.NAME\_BRA**  **FROM CHHS.dbo.EMPLOYEE**  **INNER JOIN CHHS.dbo.BRANCH ON EMPLOYEE.ID\_BRA = BRANCH.ID\_BRA** |

# **Kịch bản chứng minh tính trong suốt**

* Khi nhân viên tất cả các trạm/máy chủ muốn xem thông tin các mặt hàng trong kho, từ tất cả các trạm/máy chủ:

|  |
| --- |
| **SELECT \* FROM CHHS.dbo.SEAFOOD** |

* Khi nhân viên tất cả các trạm/máy chủ muốn xem thông tin tất cả các nhân viên trên hệ thống, từ tất cả các trạm/máy chủ:

|  |
| --- |
| **SELECT \* FROM HN2SERVER.CHHS.dbo.EMPLOYEE** |

* Nhân viên từ một trạm/máy chủ muốn xem chi tiết bất kỳ hóa đơn nào (thông qua mã hóa đơn khách hàng cung cấp) dù là có trên hệ thống mình hay không. Từ tất cả các trạm/máy chủ:

|  |
| --- |
| **IF (SELECT ID\_BRA FROM HN2SERVER.CHHS.dbo.STORAGE, HN2SERVER.CHHS.dbo.INVOICE WHERE STORAGE.ID\_STO = (SELECT ID\_STO FROM HN2SERVER.CHHS.dbo.INVOICE WHERE ID\_INV = 1005)) = 'BRA1'**  **SELECT \* FROM TB1SERVER.CHHS.dbo.INVOICE\_DETAIL WHERE ID\_INV = 1005;**  **ELSE**  **SELECT \* FROM ND2SERVER.CHHS.dbo.INVOICE\_DETAIL WHERE ID\_INV = 1005;** |

# **Kịch bản phân quyền trong hệ thống**

* Từ phía chi nhánh Thái Bình, một truy cập trái phép vào hệ thống Thái Bình để xem thông tin chi tiết một nhân viên tên Lê Thị Yến Nhi trên máy chủ (vì chỉ có trên máy chủ mới lưu thông tin chi tiết địa chỉ, sđt của tất cả nhân viên trong hệ thống):

|  |
| --- |
| **SELECT \* FROM HNSERVER.CHHS.dbo.EMPLOYEE WHERE FULLNAME = N'Lê Thị Yến Nhi'** |

* Thì phát hiện nhân viên này đang ở chi nhánh Nam Định. Truy cập trái phép này quyết định xóa tất cả thông tin nhân viên này ở máy chủ đi:

|  |
| --- |
| **DELETE FROM HNSERVER.CHHS.dbo.CUSTOMER**  **WHERE FULLNAME = N'Lê Thị Yến Nhi'** |

* + *Sau đó Show thầy xem thông báo*

# **Kịch bản xác minh thông tin khác chi nhánh**

* Một khách hàng đã từng mua hàng tại Thái Bình, đến chi nhánh Nam Định để mua đợt hàng khác, khách hàng nói với nhân viên Nam Định là đã tất cả đơn hàng với trị giá ABC trước đây ở TB và lần này muốn nhận ưu đãi.  
  Nhân viên tại TB yêu cầu mã khách hàng để kiểm tra:

|  |
| --- |
| **SELECT \* FROM HNSERVER.CHHS.dbo.INVOICE WHERE ID\_CUS = N'1001'** |

# **Kịch bản Concurrency & Commit Protocols**

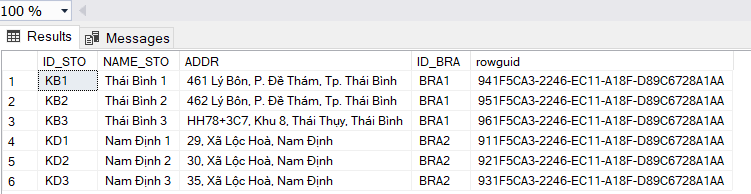
We use Serializable isolation level for all examples:

“SET TRANSACTION ISOLATION LEVEL SERIALIZABLE”

### ***Two transactions entered in a non-serializable order***

#### **+ CASE 1:** A delay

Use STORAGE table:



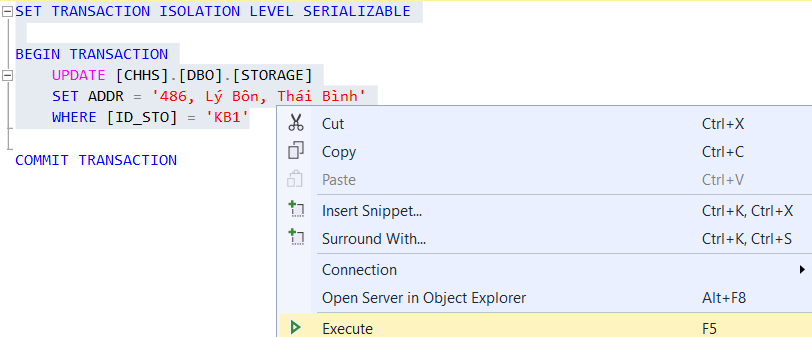
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

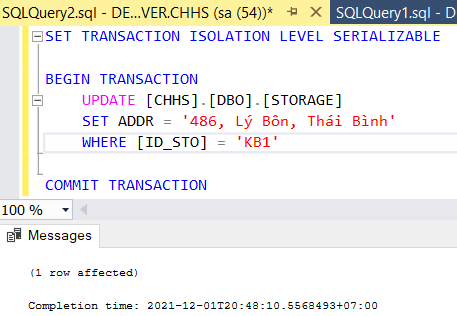
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '486, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

T2 – Show all STORAGE table

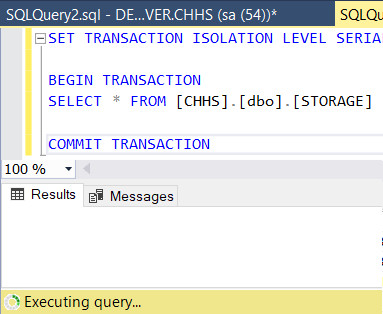
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  SELECT \* FROM [CHHS].[dbo].[STORAGE]  COMMIT TRANSACTION |

* Run T1 without committing it:



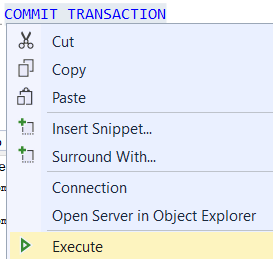


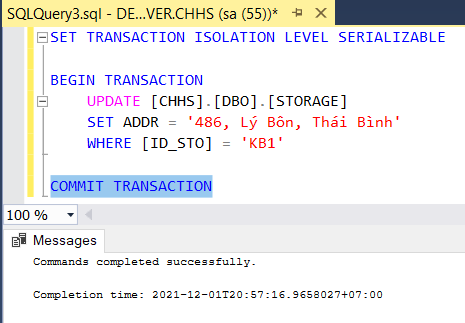
Run T2:



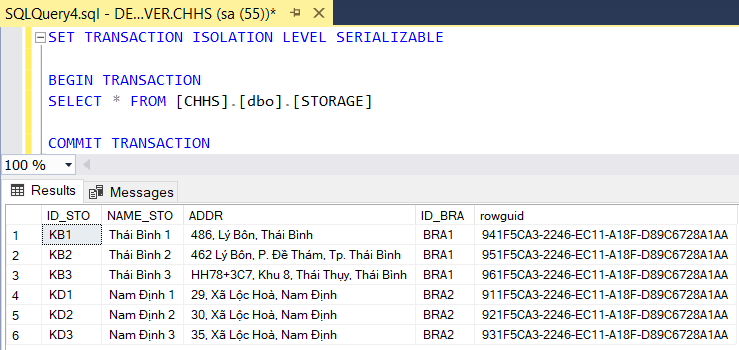
Nothing result (because of waiting for T1 to commit)

* Run COMMIT TRANSACTION:





Run T2 again:

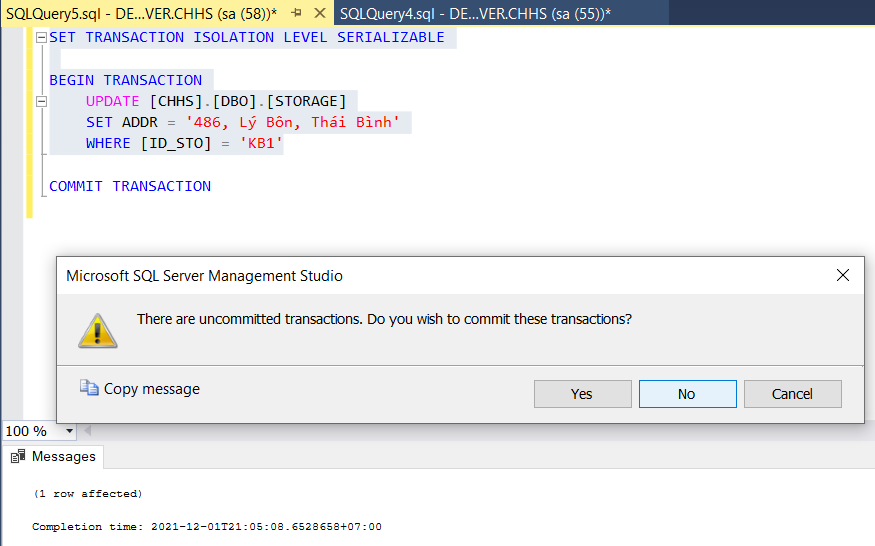


It runs immediately. Perfect! Because T1 has commited it

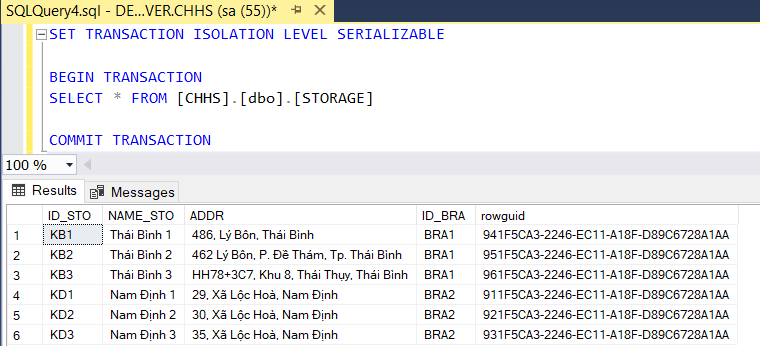
#### **+ CASE 2:** A aborted

We still use T1 and T2. But when we run T1 without commit it. And we close it:

Click No:



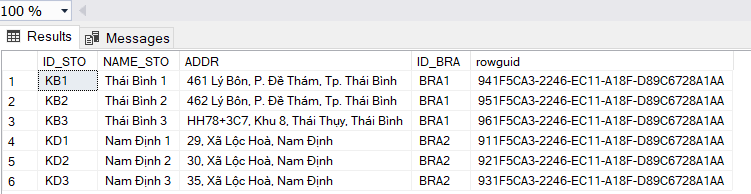
When we run T2, it’s completed immediately because the change from T1 doesn’t affect the current data ( it hasn’t yet been committed )



**Summery:** Two transactions entered in a non-serializable order will somehow be delayed, aborted, or otherwise managed so the outcome is equivalent to some serial ordering

### ***Two transactions affect on the same tables -different rows***

We still use STORAGE table:



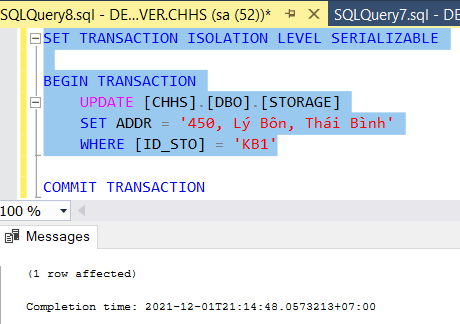
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

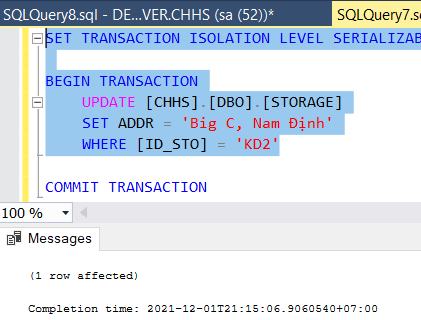
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '450, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

T2 – Update Address of Nam Định 2 Storage (ID = KD2)

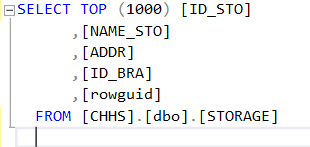
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = 'Big C, Nam Định'  WHERE [ID\_STO] = 'KD2'  COMMIT TRANSACTION |

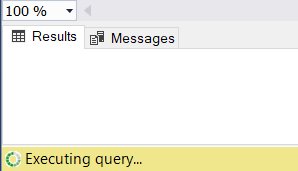
Run T1 & T2 without committing:



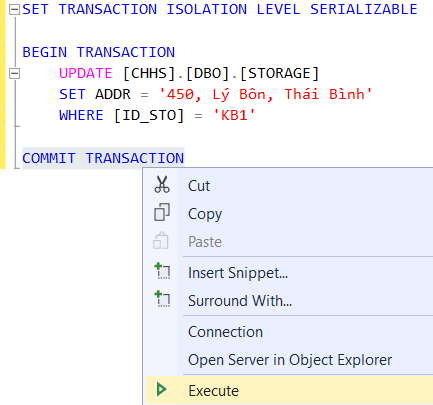


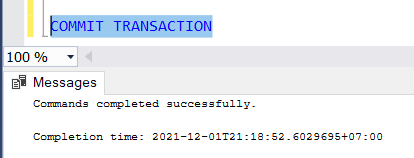
Select all STORAGE table:



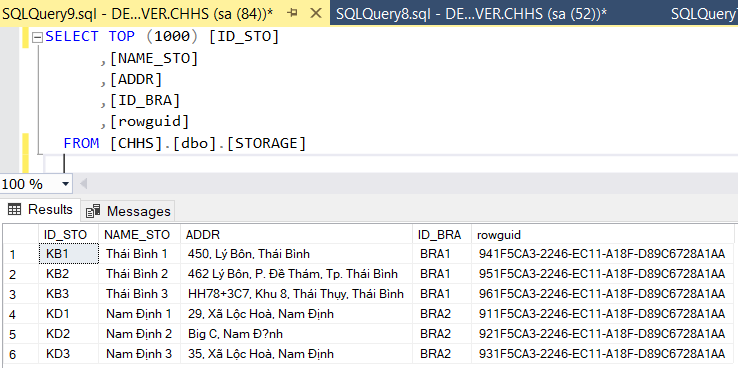


Now Commit T1 & T2:





Select all STORAGE table again:

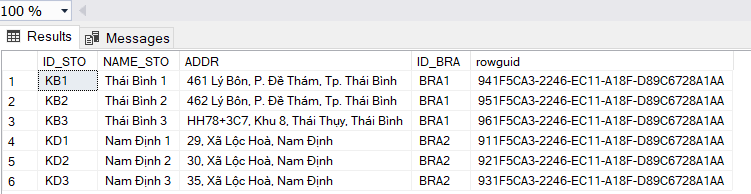


It’s completed immediately!

**Summery:** Two transactions operating on the same tables, but different rows, can execute concurrently

### ***Read/write conflict***

We still use STORAGE table:



T1 – Read STORAGE table:

|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  SELECT \* FROM [CHHS].[dbo].[STORAGE]  COMMIT TRANSACTION |

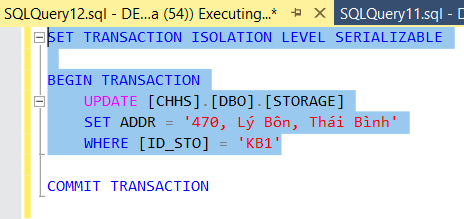
T2 – Write Address of Thái Bình 1 Storage (ID = KB1)

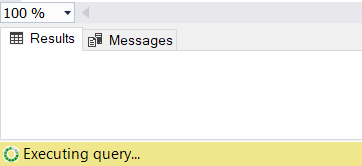
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '470, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

Run T1 without commiting it:

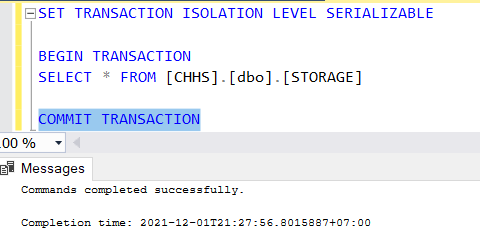


Run T2 without commiting it:

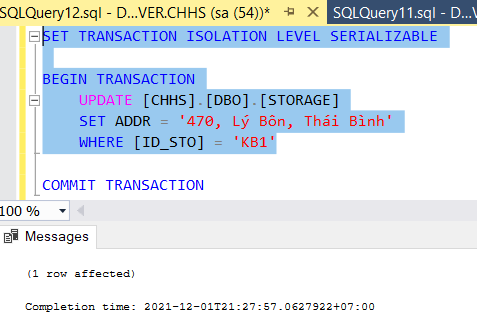




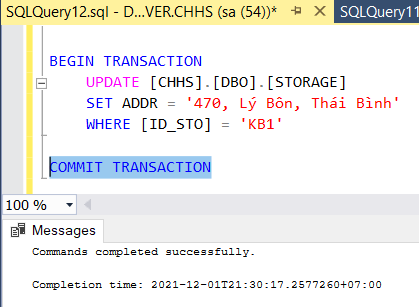
Now, Commit T1:



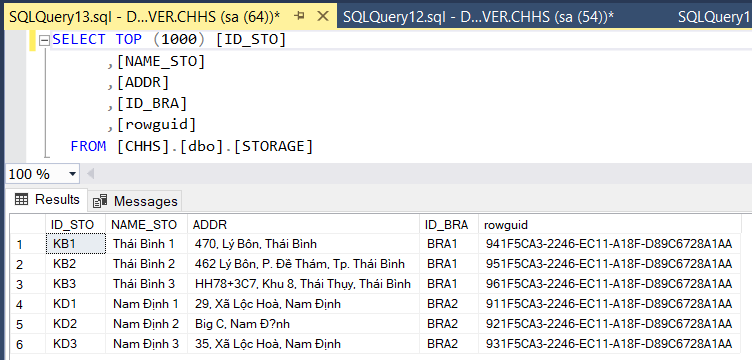
Then, move to T2 tab, we can see T2 has affected:



Now, Commit T2:



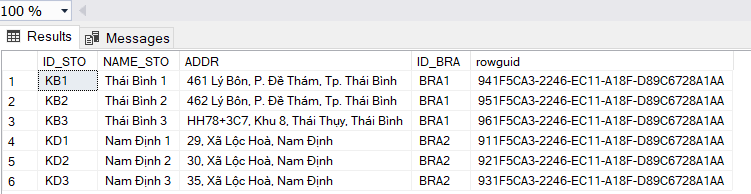
Finally, show STORAGE table:



It done!

### ***Write/write conflict***

We still use STORAGE table:



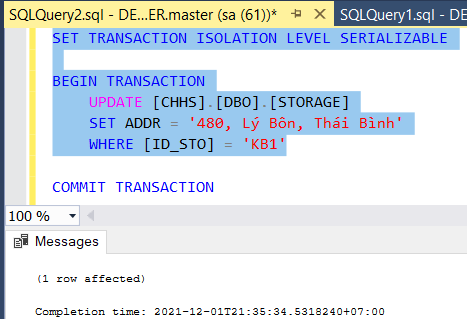
T1 – Write Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '480, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

T2 – Write another Address of Thái Bình 1 Storage (ID = KB1)

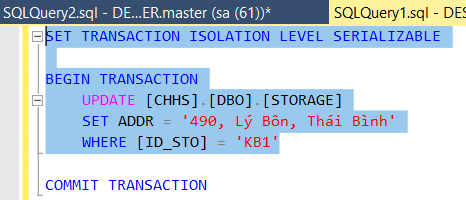
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '490, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

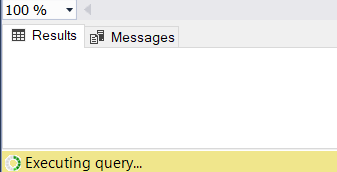
Run T1 without commiting it:



T1 is affected immediately!

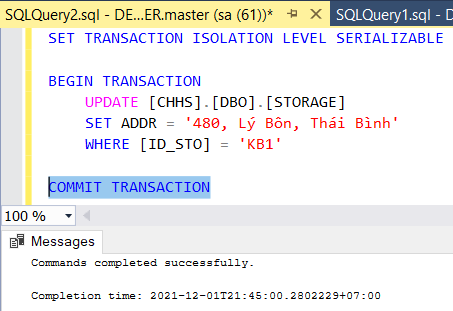
Run T2 without commiting it:



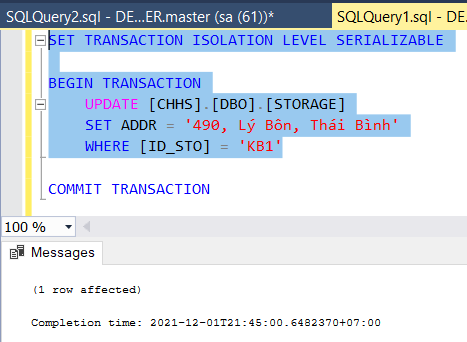


But T2 is not affected! (Because of waiting for T1)

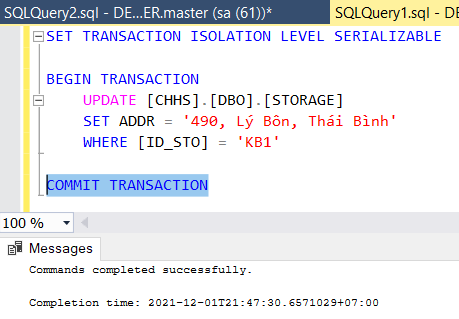
Now commiting T1:



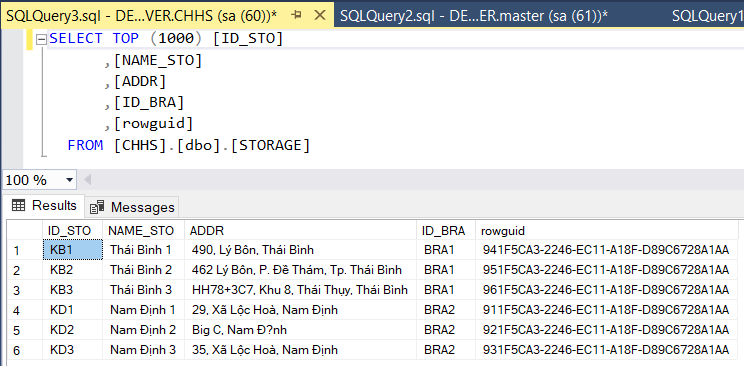
Then, move to T2 tab, we can see T2 has been affected:



Commiting T2:



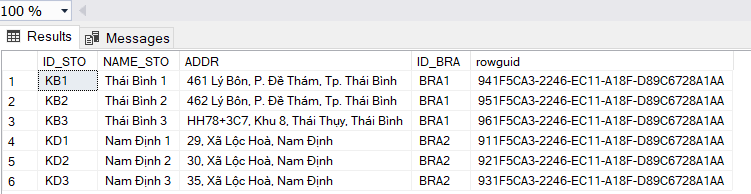
Show STORAGE table:



# **Kịch bản Distributed Failure/Recovery**

### ***Correct recovery after a commit***

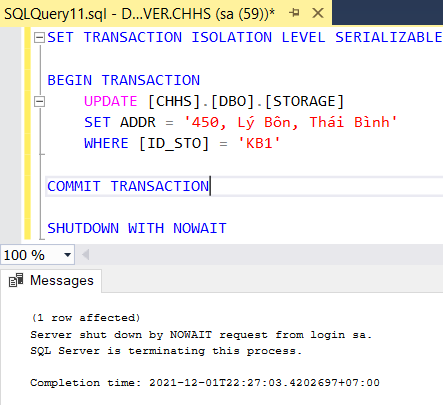
We still use STORAGE table:

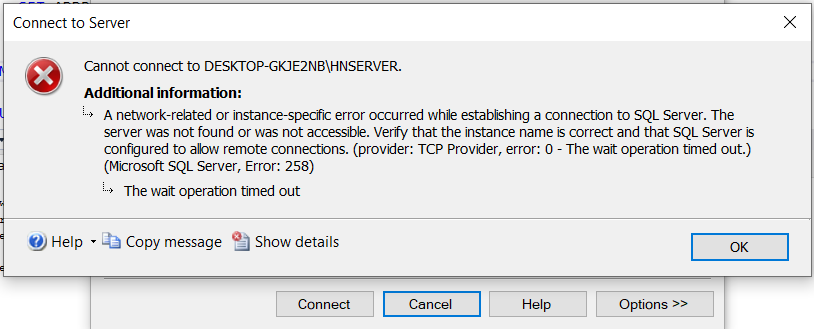


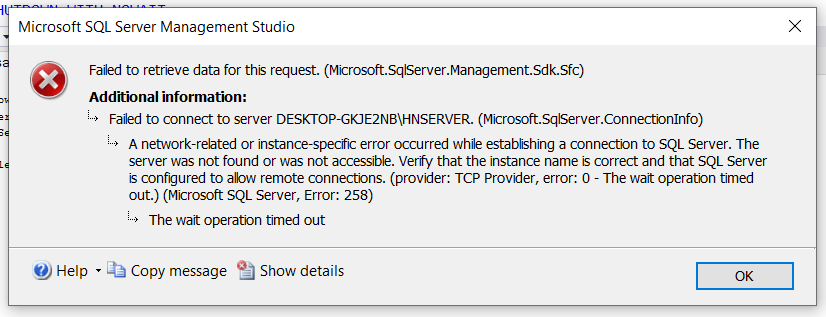
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '450, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION  SHUTDOWN WITH NOWAIT |

We run it but we shutdown service immediately

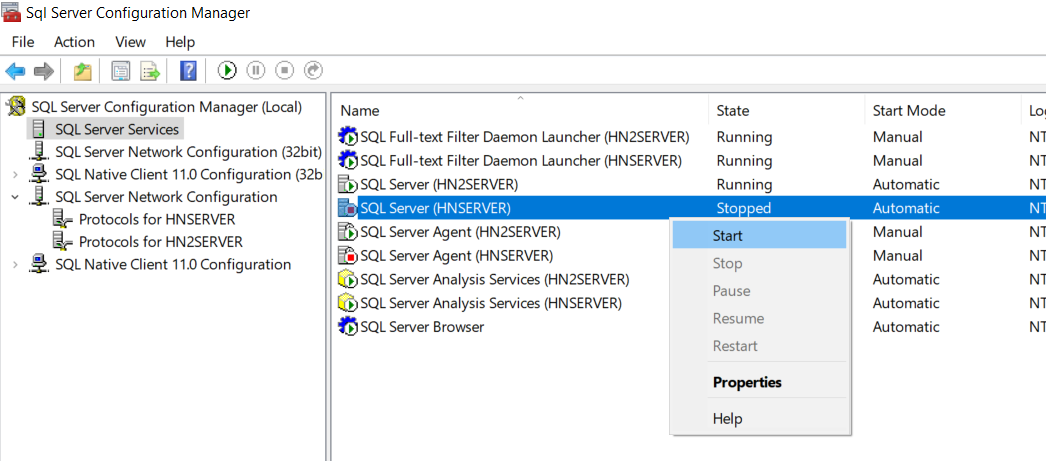


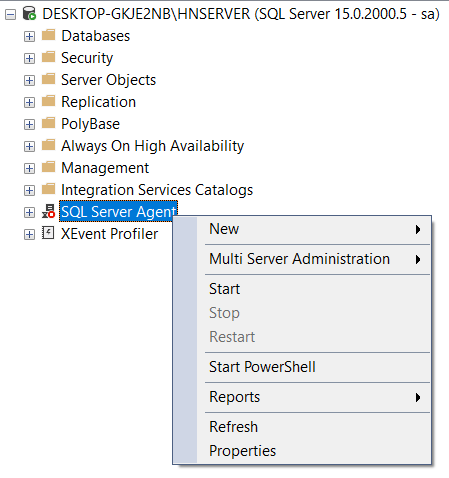




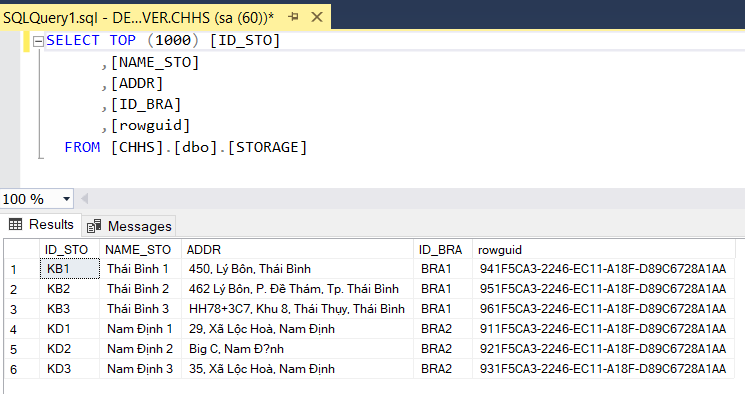
It’s stopped, we cannot refresh it, even!

We had to start it again:





Check Again:

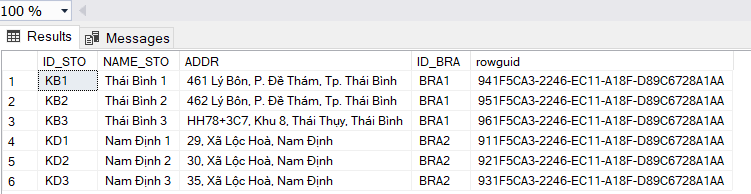


The change is committed successfully!

**Summery:** If a site crashes, the committed transactions still apply successfully

### ***Correct recovery before a commit***

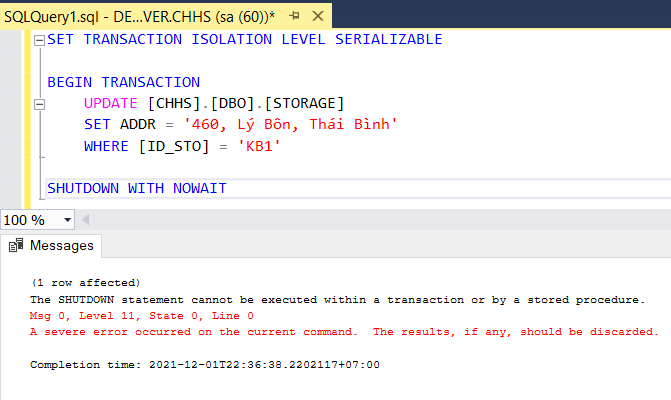
We still use STORAGE table:



T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '460, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  SHUTDOWN WITH NOWAIT |

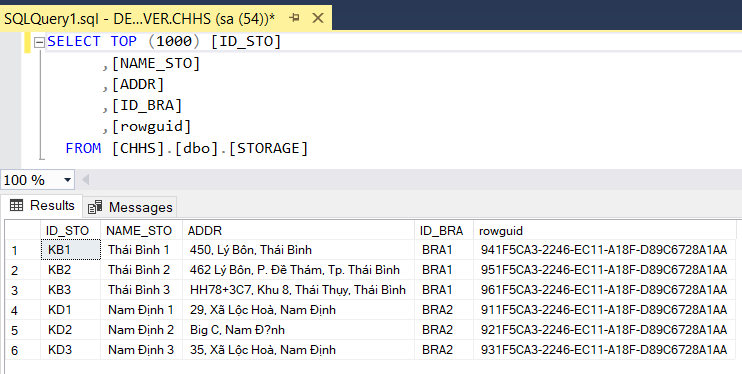
Run T1:



SQL Server won’t allow to shutdown when running a uncommitted transaction

Now, we turn of SQL server manually

And open it again and check STORAGE table:

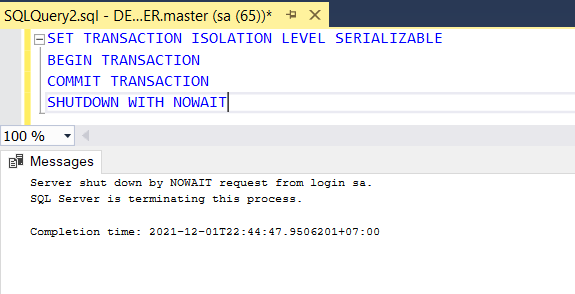


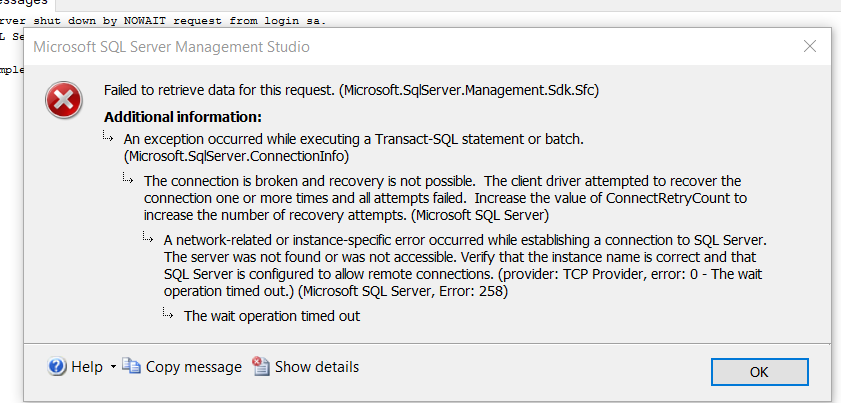
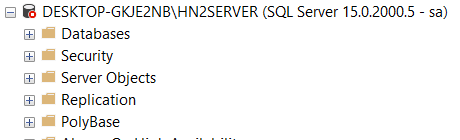
Nothing is changed!!

**Summery:** All the uncommitted transactions are aborted in a slave server if there is a crash in that server, which makes no changes to the master serve

### ***A server site is crashed***

We shutdown HN2SERVER:



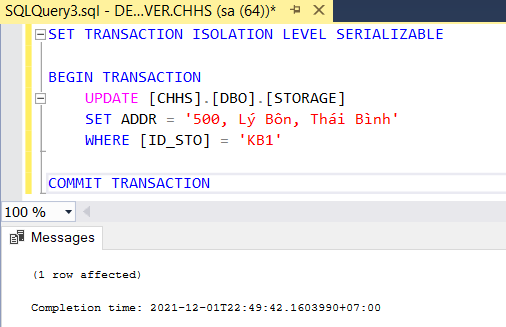
 

Now we will try to send transaction between other servers, suppose the master server update the STORAGE table:

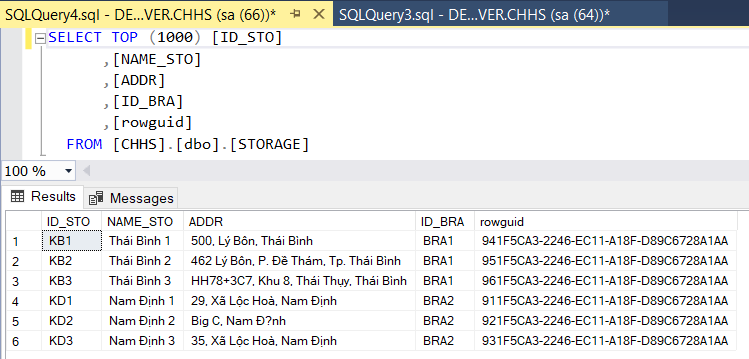
T2 – Update Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '500, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

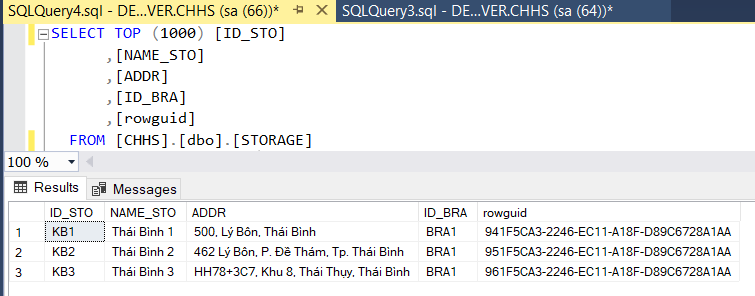
Run T2 on Ha Noi Server (master server):



Check STORAGE table on Ha Noi server:



Check STORAGE table on Thai Binh server:



It still be ok! It’s updated!

**Summery:** Even if a server site is crashed, as long as the query is not related to the crashed one, the remaining sites are working properly