**POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY**

**INFORMATION TECHNOLOGY DEPARTMENT**



****

**COURSEWORK**

**Distributed Database Systems**

**Discussion: BUILDING A DISTRIBUTED DATABASE SYSTEM TO MANAGE SEAFOOD COMPANY**

|  |  |
| --- | --- |
| ***Lecturer:*** | **PhD. Nguyen Dinh Hoa** |
| ***Team:*** | **08** |
| ***Class:*** | **E18CQCN01-B** |

**Hanoi, Nov 2021**

**A picture containing text, clipart

Description automatically generated**

**Team 08**

|  |  |
| --- | --- |
| **Full name** | **Student num.** |
| Nguyễn Thành Đạt (Leader) | B18DCCN141 |
| Hà Quốc Huy | B18DCCN263 |
| Trần Huy Nam | B18DCDT171 |

**POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**



**Hanoi - 2021**

**COURSEWORK**

**Distributed Database Systems**

**Contents**

[**I.** **Real Worlf Scenario** 6](#_Toc89291850)

[1. Importance of the project 6](#_Toc89291851)

[2. Benefits of using distributed system 6](#_Toc89291852)

[**II.** **Analysis** 7](#_Toc89291853)

[**1.** **The main functions of the system** 7](#_Toc89291854)

[• Data management of materials 7](#_Toc89291855)

[• Manage warehouse information 7](#_Toc89291856)

[• Manage employee information 7](#_Toc89291857)

[• Manage warehouse import and export information 7](#_Toc89291858)

[• Manage customer information 7](#_Toc89291859)

[• Provider information management 7](#_Toc89291860)

[**2.** **Function of each position** 7](#_Toc89291861)

[**2.1.** **Function at workstations (branch warehous: Thai Binh, Nam Dinh)** 7](#_Toc89291862)

[**2.2.** **Functions at the server (headquater: Hanoi)** 7](#_Toc89291863)

[**2.3.** **Decentralize system object** 7](#_Toc89291864)

[**3.** **Database analysis** 8](#_Toc89291865)

[**3.1.** **Linked enity model** 8](#_Toc89291866)

[**3.2. Relationship** 9](#_Toc89291867)

[**3.3.** **Frequency table of accessing locations** 9](#_Toc89291868)

[**III.** **Design** 10](#_Toc89291869)

[**1.** **Overall design** 10](#_Toc89291870)

[**2.** **Design database** 11](#_Toc89291871)

[**2.1.** **List of tables** 11](#_Toc89291872)

[**2.2.** **Diagram** 15](#_Toc89291873)

[**3.** **Fragmentation** 15](#_Toc89291874)

[3.1. Horizontal fragmentation of data 15](#_Toc89291875)

[3.2. Vertical fragmentation of data 18](#_Toc89291876)

[**IV.** **Setting** 18](#_Toc89291877)

[1. Install SQL server 18](#_Toc89291878)

[2. Setting SMSS 28](#_Toc89291879)

[3. Setting Radmin VPN 29](#_Toc89291880)

[**V.** **Configuration** 32](#_Toc89291881)

[**1.** **Create Shared Folder** 32](#_Toc89291882)

[**2.** **Setting Firewall** 35](#_Toc89291883)

[**3.** **Configuration IP** 39](#_Toc89291884)

[**VI.** **Database settings and fragmentation** 43](#_Toc89291885)

[2.2 Vertical fragmentation 58](#_Toc89291886)

[**VII.** **Connect, Decentralize – ensure transparency** 62](#_Toc89291887)

[1. Configure the “sa” account 62](#_Toc89291888)

[2. Create a read-only account 66](#_Toc89291889)

[3. Connect 69](#_Toc89291890)

[**VIII.** **Distributed CSDL** 72](#_Toc89291891)

[Create Subscriptions 72](#_Toc89291892)

[**IX.** **Query** 77](#_Toc89291893)

[1. Show all Customer from Ha Noi 2 (HN2SERVER) from another branch 77](#_Toc89291894)

[2. Show Name and Numer of top 10 Customer from Nam Dinh (ND2SERVER) 78](#_Toc89291895)

[3. Show top 10 Employee at Thai Binh from HN2SERVER 78](#_Toc89291896)

[4. Insert data from Ha Noi (HNSERVER) and test the synchronization in Ha Noi 2 (HN2SERVER) 79](#_Toc89291897)

[5. Update Customer from Ha Noi (HNSERVER) and test the synchronization 80](#_Toc89291898)

[6. Find out a customer whose number phone is 0988585568 to test the Connection 82](#_Toc89291899)

[7. Test permission (Only Read) 82](#_Toc89291900)

[8. Show all Invoice and Sort by Total Price 83](#_Toc89291901)

[**X.** **Concurrency & Commit Protocols** 83](#_Toc89291902)

[**1.** **Transaction** 83](#_Toc89291903)

[**2.** **Commit Protocols** 84](#_Toc89291904)

[2.1. One-phase Commit 84](#_Toc89291905)

[2.2. Two-Phase Commit 85](#_Toc89291906)

[**3.** **Concurrency control** 86](#_Toc89291907)

[3.1. Concurrency Problem Types 86](#_Toc89291908)

[3.2. Transaction isolation level 86](#_Toc89291909)

[3.3. Two Phase Locking – 2PL 87](#_Toc89291910)

[3.4. Strict strong Two Phase Locking (SS2PL) 87](#_Toc89291911)

[**4.** **Practise** 87](#_Toc89291912)

[4.1. Two transactions entered in a non-serializable order 88](#_Toc89291913)

[4.2. Two transactions affect on the same tables -different rows 92](#_Toc89291914)

[4.3. Read/write conflict 95](#_Toc89291915)

[4.4. Write/write conflict 99](#_Toc89291916)

[**5.** **Distributed Failure/Recovery** 102](#_Toc89291917)

[5.1. Types of failures: 102](#_Toc89291918)

[5.2. Correct recovery after a commit 103](#_Toc89291919)

[5.3. Correct recovery before a commit 107](#_Toc89291920)

[5.4. A server site is crashed 108](#_Toc89291921)

[**References** 112](#_Toc89291922)

# **Real Worlf Scenario**

## **Importance of the project**

The storage of large amounts of information is a dilemma in the process of operating retail chains such as clothing, food, etc., especially for seafood store management chains. A system that needs to store a lot of information such as: employee information, product information, ... causes difficulties in the process of managing, comparing information and prone to errors. In the case of a seafood store, checking the amount of items in stock is necessary to make decisions in import and export, activities that directly affect the financial resources of the store. Thus, the bookkeeping method has many limitations. To facilitate management, it is necessary to build a digital system and digitize all necessary information to be stored into one system.

With the high consumption demand of people throughout the provinces, it is necessary to open more branches. At this time, data management by a centralized basis will reveal many weaknesses. Since the data is stored at only one main server, then when the branches need to retrieve information, they will have to send requests there, waiting for a response, which takes a long time. In addition, when new data is updated in large quantities, the server must process a lot, take time and may lose information.

In fact, each branch will have the need to manage its own type of information and data, because the amount of data at each branch is very large, so it is reasonable to use a distributed database system here.

## **Benefits of using distributed system**

* **Staff:** Easily check goods information. Import and export management suppliers
* **Economical:** Reduce operating costs, save costs on network, maintenance, inspection and data recovery as well as time to fulfil requests
* **Stores:** Make the management simple, fast, convenient and improve work efficiency

# **Analysis**

## **The main functions of the system**

## • Data management of materials

## • Manage warehouse information

## • Manage employee information

## • Manage warehouse import and export information

## • Manage customer information

## • Provider information management

## **Function of each position**

### ***Function at workstations (branch warehous: Thai Binh, Nam Dinh)***

* Manage information of object belonging to the branch: add, delete detail of branch warehouse, employees, customers, raw materials, inventory, import-export.
* Statistics function by monthly revenue

### ***Functions at the server (headquater: Hanoi)***

• The entire function of the branch,

• Manage branch information: can add, edit, delete detailed information of branches and warehouses belonging to branches,

• Manage material information: Add, edit, delete information of goods, raw materials,

• Statistical reporting function: revenue statistics, import-export statistics, etc.

### ***Decentralize system object***

• Management staff (at the server): can view, add, edit, delete all data information

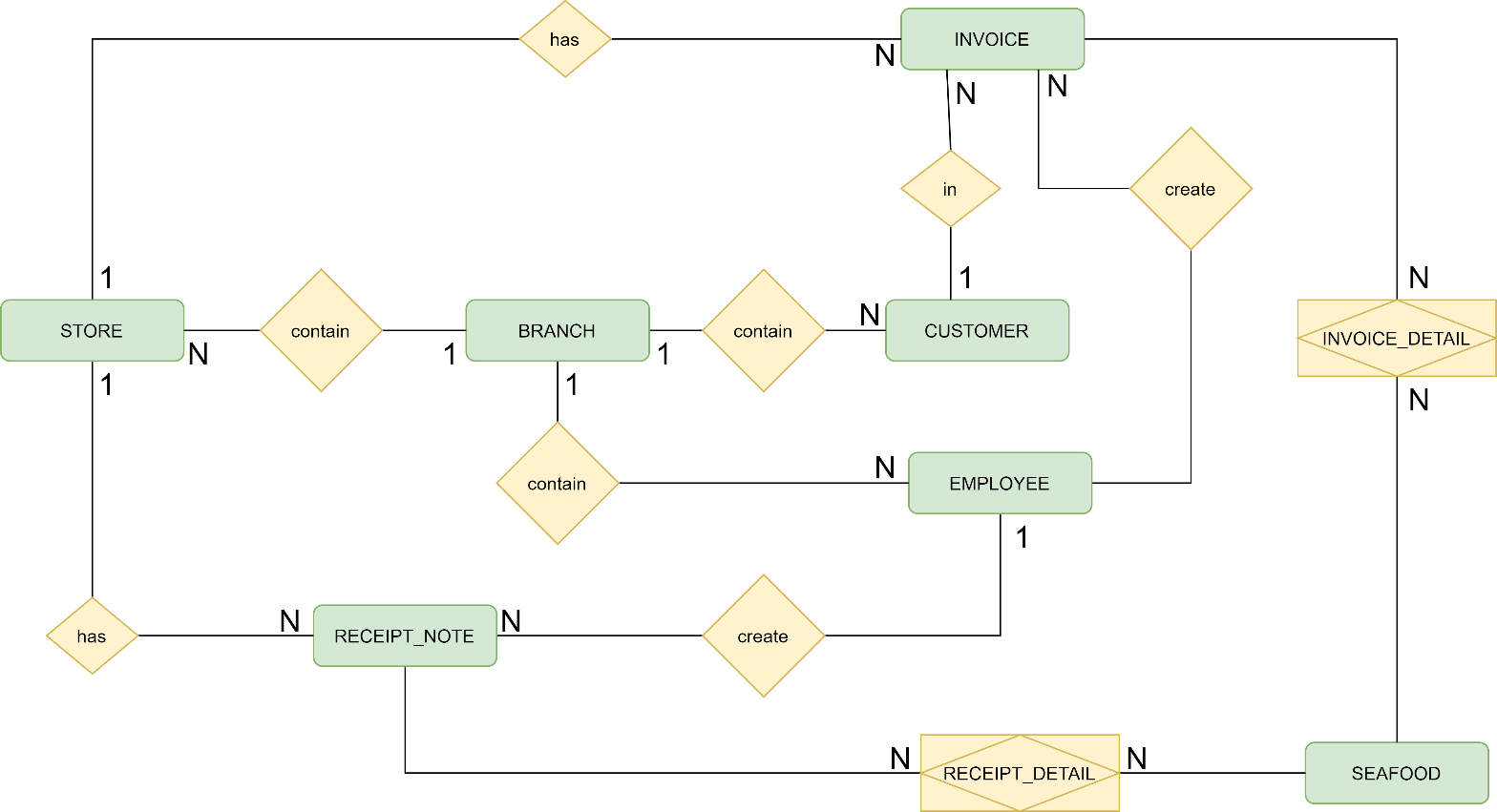
• Staff at the branch:

o Information can be viewed at the branch: branch information, branch warehouse information, customers, employees, raw materials, inventory, import-export at the branch,

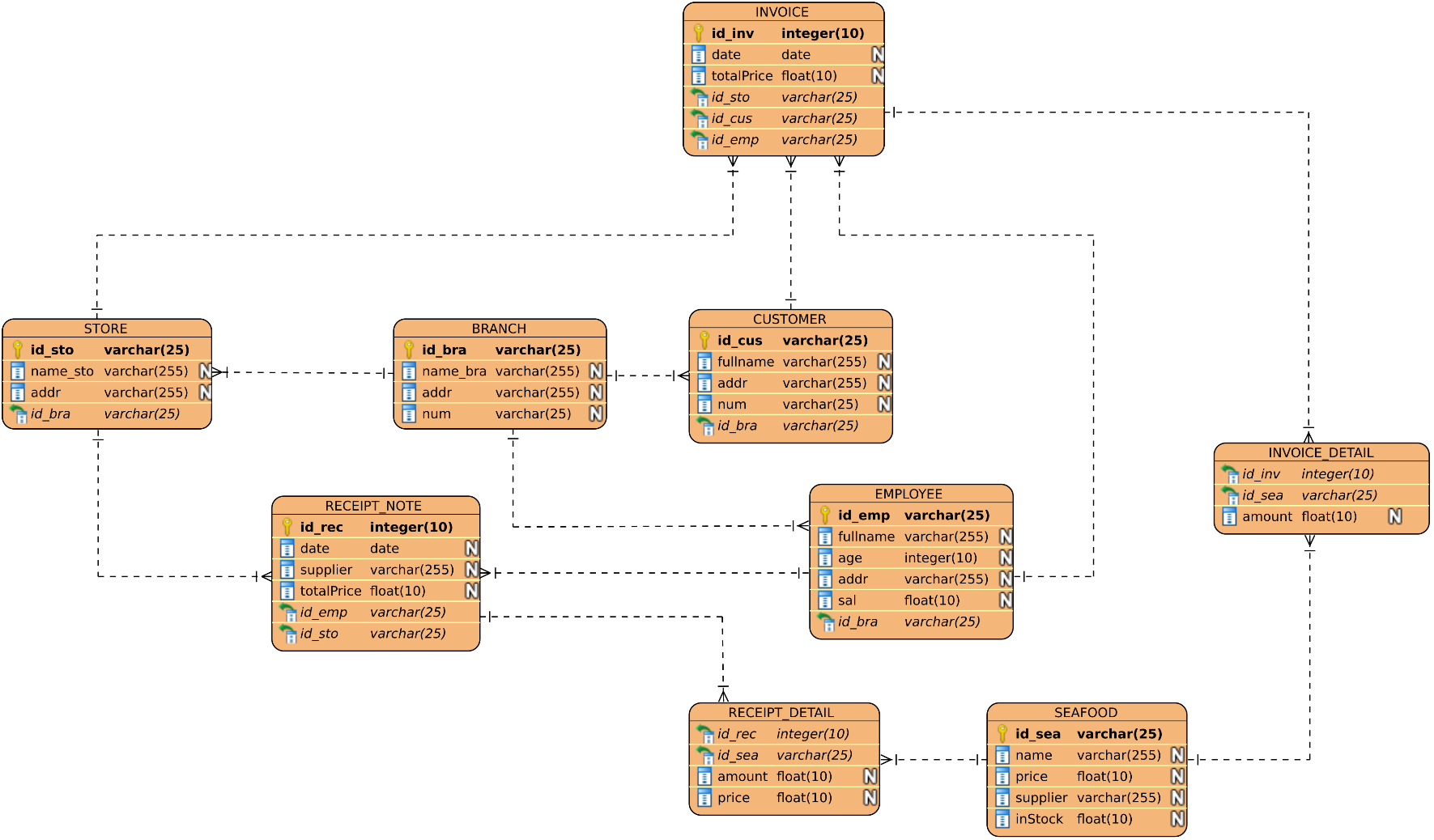
o Can add, edit, delete import and export information, employees, customers at the branch.

## **Database analysis**

### ***Linked enity model***



### ***3.2. Relationship***

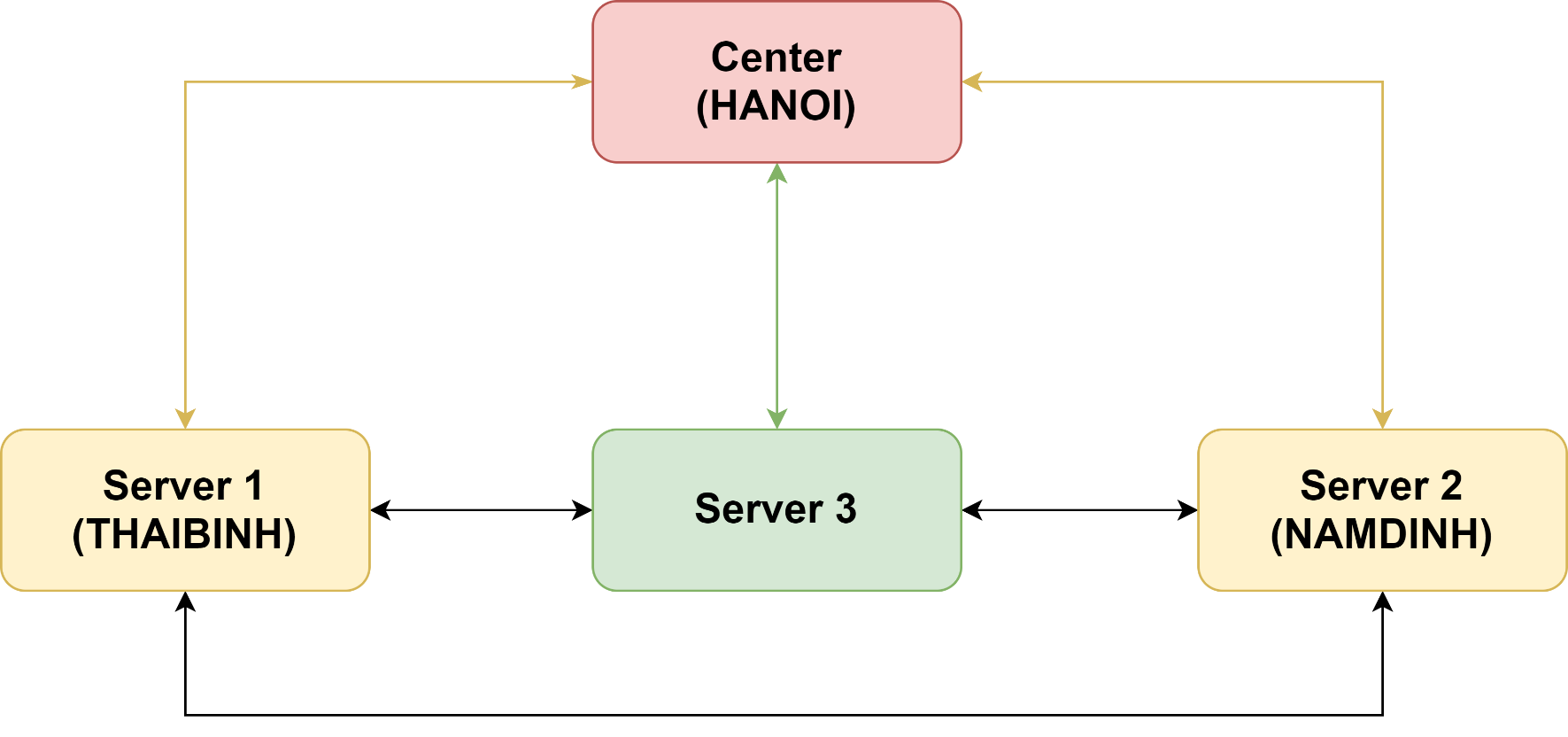


### ***Frequency table of accessing locations***

|  |  |  |
| --- | --- | --- |
| **Entity** | **Headquater** | **Branch** |
| BRANCH | H.R, L.CUD | L.R |
| STORAGE | H.R, L.CUD | H.R, L.CUD |
| CUSTOMER | H.CR, L.UD | H.CRU, L.D |
| EMPLOYEE | H.CRUD | H.CRUD |
| SEAFOOD | H.CRUD | H.RU |
| INVOICE | H.RU | H.CRU |
| INVOICE\_DETAIL | H.RU | H.CRU |
| RECEIPT\_NOTE | H.CR | H.R |
| RECEIPT\_DETAIL | H.CR | H.R |
| C (Create): Add data  R (Read): Read  U (Update): Edit data  D (Delete): Delete data | | H (High): High frequency  L (Low): Low frequency | |

# **Design**

## **Overall design**



Seafood company has a head office in Hanoi and 2 branches in Thai Binh and Nam Dinh

Distribute the database CTHS (Seafood Company) into 3:

* **Server 1 in Thai Binh:** contains information about employees, vouchers/invoices, raw materials and data generated at Thai Binh branch
* **Server 2 in Nam Dinh:** contains information about employees, vouchers/invoices, raw materials and data generated at Nam Dinh branch
* **Server 3 in Hanoi 2:** contains employee information, raw materials, suppliers, coupons/invoices, warehouse information of both branches.
* **Server in Hanoi:** contains employee information, raw materials, suppliers, coupons/ invoices, warehouse information of both branches.
* Deployment management software with one server (manager) and 3 workstations. The server aggregates data and coordinates data between substation
* Station 1, Station 2, Station 3 are located in different places, in stations containing different fragmented data, operating under the same system, linked together via communication network/intranet
* Each station is independent of the other, having a data connection with each other to request data from another station. All 3 stations have a connection to the server, when there is a change in data from the 3 stations, they are instantly synchronized to the server.
* ***Head office:*** Server (Hanoi)

- **Server user:** Restaurant chain owner

- **Data:** employee information, suppliers, raw materials, vouchers/invoices, warehouses

- **Input data:** is entered at the server or sent from the workstations

- **Output data:** saved at the server, and updated at the workstations

* ***Workstation:*** (1) Thai Binh, (2) Nam Dinh, (3) Ha Noi 2

**- User:** Manager at restaurant branches

**- Data:** employee information, material information, vouchers/invoices

**- Input data:** sent down from the server or manually entered by the employee.

**- Output data:** saved at the workstation and updated to the server's database.

## **Design database**

### ***List of tables***

**BRANCH TABLE**

Contains information about the company's branches

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Constraint** |
| id\_bra | nchar(25) | Primary key |
| name\_bra | nvarchar(255) | Unique |
| addr | nvarchar(255) |  |
| num | nchar(25) |  |

**STORE TABLE**

Contains information related to the repositories

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Constraint** |
| id\_sto | nchar(25) | Primary key |
| name\_sto | nvarchar(255) | Unique |
| addr | nvarchar(255) |  |
| id\_bra | nchar(25) | Foreign Key, ON UPDATE CASCADE |

**CUSTOMER TABLE**

Contains information related to customers

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_cus | nchar(25) | Primary key |
| fullname | nvarchar(255) |  |
| addr | nvarchar(255) |  |
| num | nchar(25) |  |
| id\_bra | nchar(25) | Foreign Key, ON UPDATE CASCADE |

**EMPLOYEE TABLE**

Contrains emplyee information

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_emp | char(25) | Primary key |
| fullname | nvarchar(255) |  |
| age | int |  |
| addr | nvarchar(255) |  |
| sal | float | > 5,000,000.0 |
| id\_bra | nchar(25) | Foreign Key, ON UPDATE CASCADE |

**SEAFOOD TABLE**

Contains information related to materials – products

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_sea | char(25) | Primary key |
| name\_sea | nvarchar(255) |  |
| price | float |  |
| supplỉer | nvarchar(255) |  |
| inStock | float | > 0.0 |

**INVOICE TABLE**

Contains invoice related information

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_inv | int | Primary key |
| Time\_date | smalldatetime | DEFAULT(GETDATE()) |
| totalPrice | float |  |
| id\_cus | nchar(25) | Foreign Key |
| id\_emp | nchar(25) | Foreign Key |
| id\_sto | nchar(25) | Foreign Key, ON UPDATE CASCADE |

**INVOICE\_DETAIL TABLE**

Contains information related to invoice details

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_inv | int | Primary key, Foreign Key, ON UPDATE CASCADE |
| id\_sea | char(25) | Primary key, Foreign Key, ON UPDATE CASCADE |
| amount | float | > 0.0 |

**RECEIPT\_NOTE TABLE**

Contains information related to the entry form

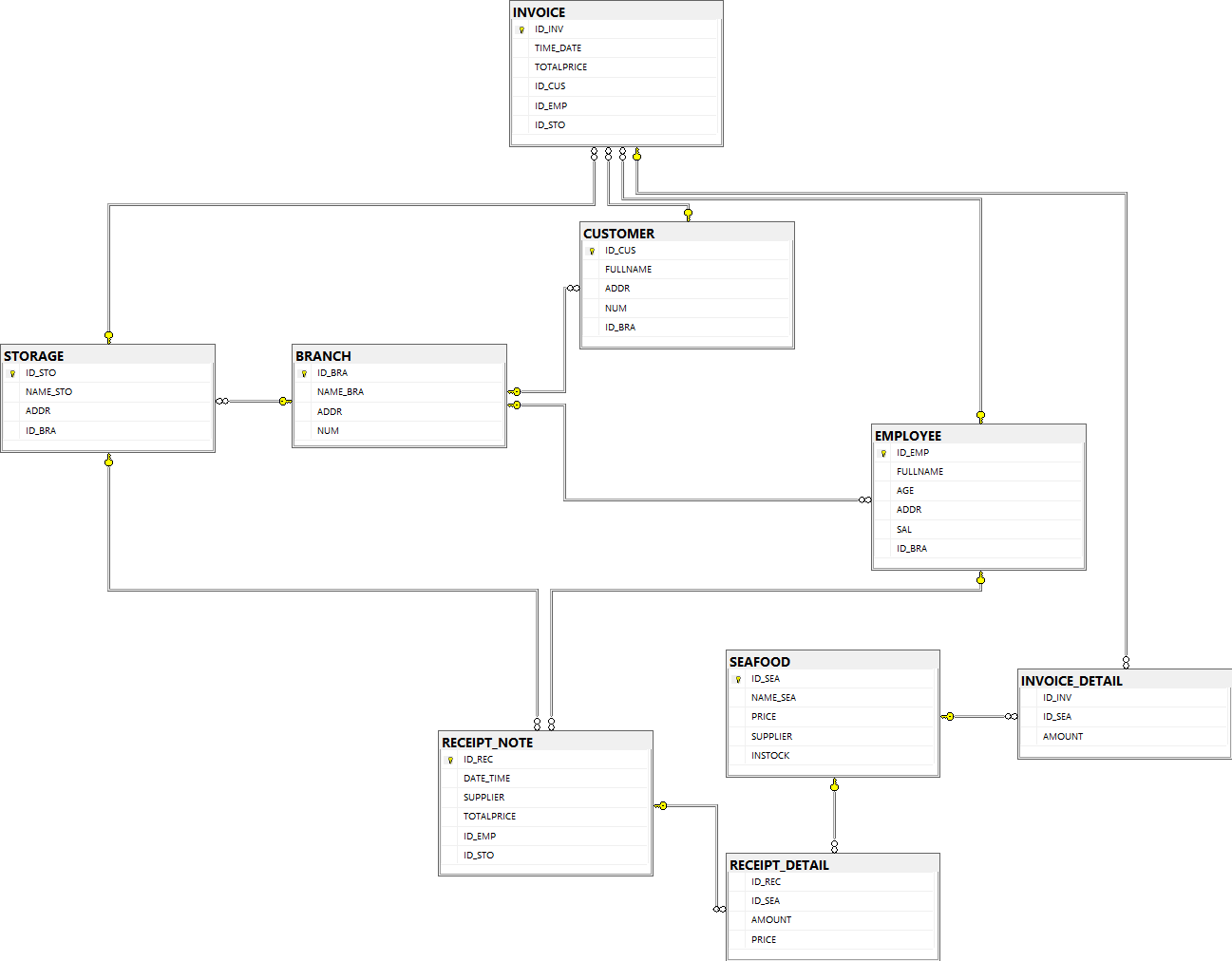
|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_rec | int | Primary key |
| date\_time | date | DEFAULT(GETDATE()) |
| source | nvarchar(255) |  |
| totalPrice | float |  |
| id\_emp | char(25) | Foreign Key |
| id\_sto | nchar(25) | Foreign Key, ON UPDATE CASCADE |

**RECEIPT\_DETAIL TABLE**

Contains information related to entry slip details

|  |  |  |
| --- | --- | --- |
| **Entity** | **Type data** | **Contraint** |
| id\_rec | int | Primary key, Foreign Key, ON UPDATE CASCADE |
| id\_sea | char(25) | Primary key, Foreign Key, ON UPDATE CASCADE |
| Amount | float | > 0.0 |
| price | float |  |

### ***Diagram***



Because the branches can manage the information in a complete table like in the main system. It is recommended to use horizontal fragmentation to disperse the data so that the physical structure of the data tables at the stations is the same. System data structure:

## **Fragmentation**

### ***Horizontal fragmentation of data***

BRANCH tables will be used by all sites, but can only be updated, edited or deleted on the master server.

The tables EMPLOYEE, STORAGE, CUSTOMER, INVOICE, INVOICE\_DETAIL, SEAFOOD, RECEIPT\_NOTE, RECEIPT\_DETAIL will be used separately at each site.

Primitive horizontal fragmentation and derived fragmentation divide the overall relationship into 3 pieces located at 3 branches:

• Location 1: Server located in Hanoi

• Location 2: Workstation 1 is located in Thai Binh

• Location 3: Workstation 2 is located in Nam Dinh

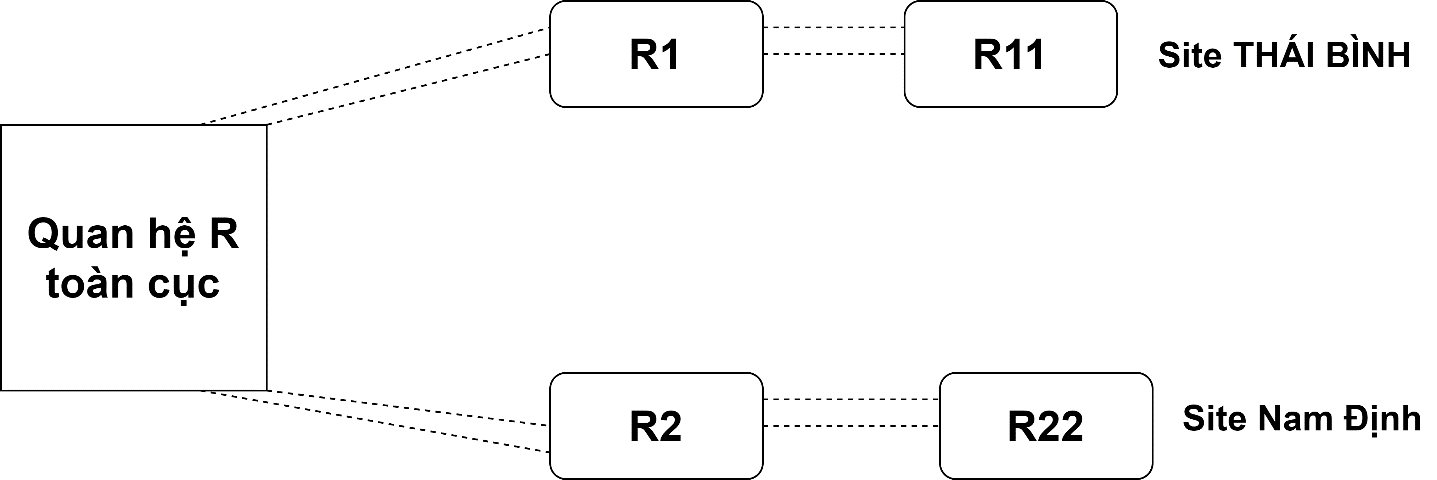
For fragmentation, we choose the overall relation BRANCH as the criterion for horizontal fragmentation into 2 located at 2 locations, then based on those fragments to defragment derivatives to the remaining relations.

#### **Fragmentation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Piece** | **Server** | **Primitive horizontal** | **Derived horizontal fragmentation** |
| 1 | Thai Binh(1) | Fragmentation table: BRANCH  BRANCH1 = | Fragmentation table: CUSOMTER  CUSTOMER1 =  CUSTOMER ▷◁ BRANCH1 |
|  | Fragmentation table: EMPLOYEE  EMPLOYEE1 =  EMPLOYEE ▷◁ BRANCH1 |
| Fragmentation table: STORAGE  STORAGE1 =  STORAGE ▷◁ BRANCH1 |
| Fragmentation table: INVOICE  INVOICE1 =  INVOICE ▷◁ STORAGE1 |
| Fragmentation table: INVOICE\_DETAIL  INVOICE\_DETAIL1 =  INVOICE\_DETAIL ▷◁ INVOICE1 |
| Fragmentation table: RECEIPT\_NOTE RECEIPT\_NOTE1 =  RECEIPT\_NOTE ▷◁ STORAGE1 |
| Fragmentation table: RECEIPT\_DETAIL RECEIPT\_DETAIL1 =  RECEIPT\_DETAIL ▷◁ RECEIPT\_NOTE1 |
| 2 | Nam Định (2) | Fragmentation table: BRANCH  BRANCH2 = | Fragmentation table: CUSOMTER  CUSTOMER2 =  CUSTOMER ▷◁ BRANCH2 |
|  | Fragmentation table: EMPLOYEE  EMPLOYEE2 =  EMPLOYEE ▷◁ BRANCH2 |
| Fragmentation: STORAGE  STORAGE2 =  STORAGE ▷◁ BRANCH2 |
| Fragmentation table: INVOICE  INVOICE2 =  INVOICE ▷◁ STORAGE2 |
| Fragmentation table: INVOICE\_DETAIL  INVOICE\_DETAIL2 =  INVOICE\_DETAIL ▷◁ INVOICE2 |
| Fragmentation table: RECEIPT\_NOTE RECEIPT\_NOTE2 =  RECEIPT\_NOTE ▷◁ STORAGE2 |
| Fragmentation table: RECEIPT\_DETAIL RECEIPT\_DETAIL2 =  RECEIPT\_DETAIL ▷◁ RECEIPT\_NOTE2 |

#### **Mapping Schema**

At the server, the global relationship via horizontal fragmentation is split into 2 fragments CHHS\_BRA1 and CHHS\_BRA2. Each piece has only 1 copy at a certain tramh



## **Vertical fragmentation of data**

Vertical fragmentation of CUSTOMER, EMPLOYEE, BRANCH tables

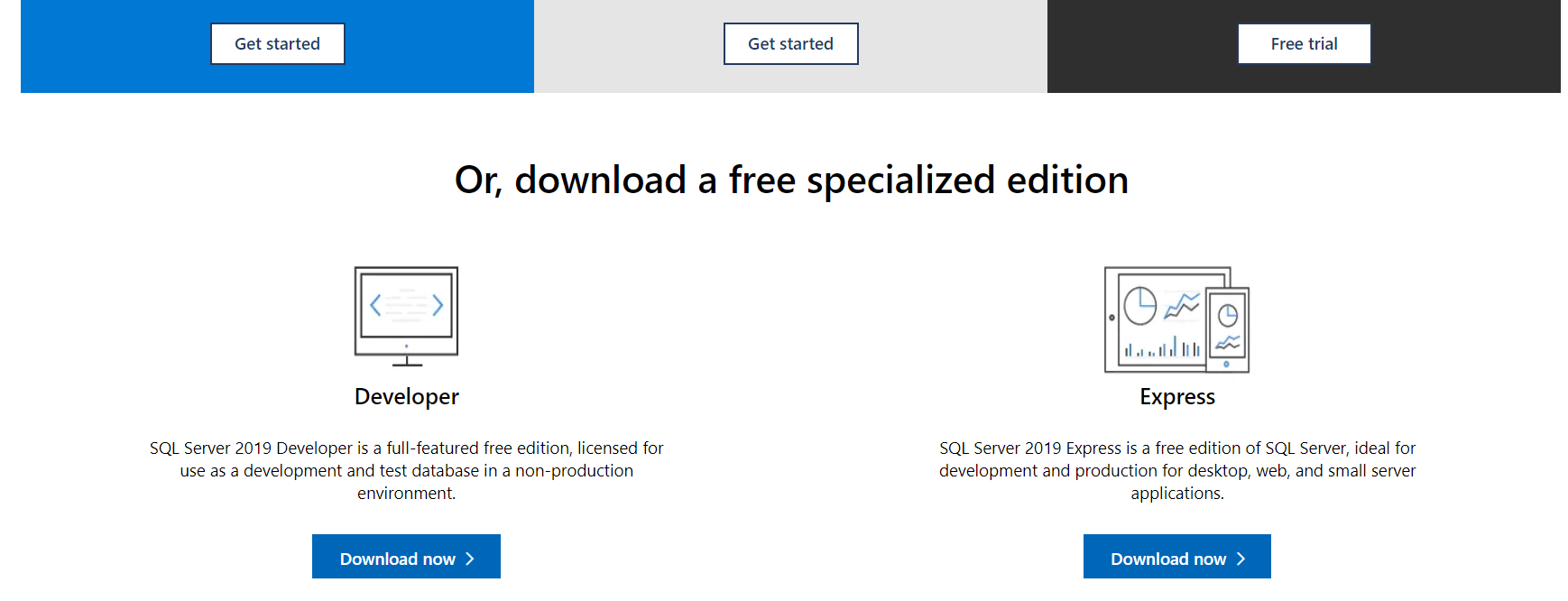
# **Setting**

## **Install SQL server**

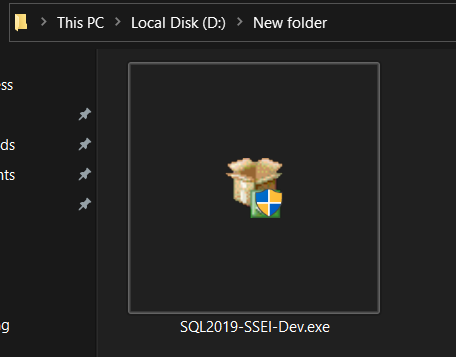
Installed on all machines

All machines need to install SQL server 2019.

Download here: [**https://www.microsoft.com/en-us/sql-server/sql-server-downloads**](https://www.microsoft.com/en-us/sql-server/sql-server-downloads)



**Note:** download developer or enterprise version, do not download Express version

After downloading the file:

We use the developer here

Choose CUSTOM



The following steps are default or can choose from

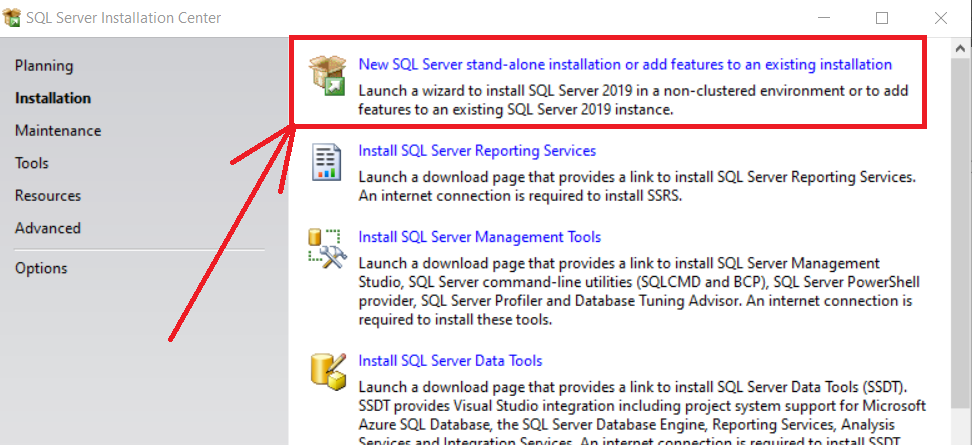
After setting:



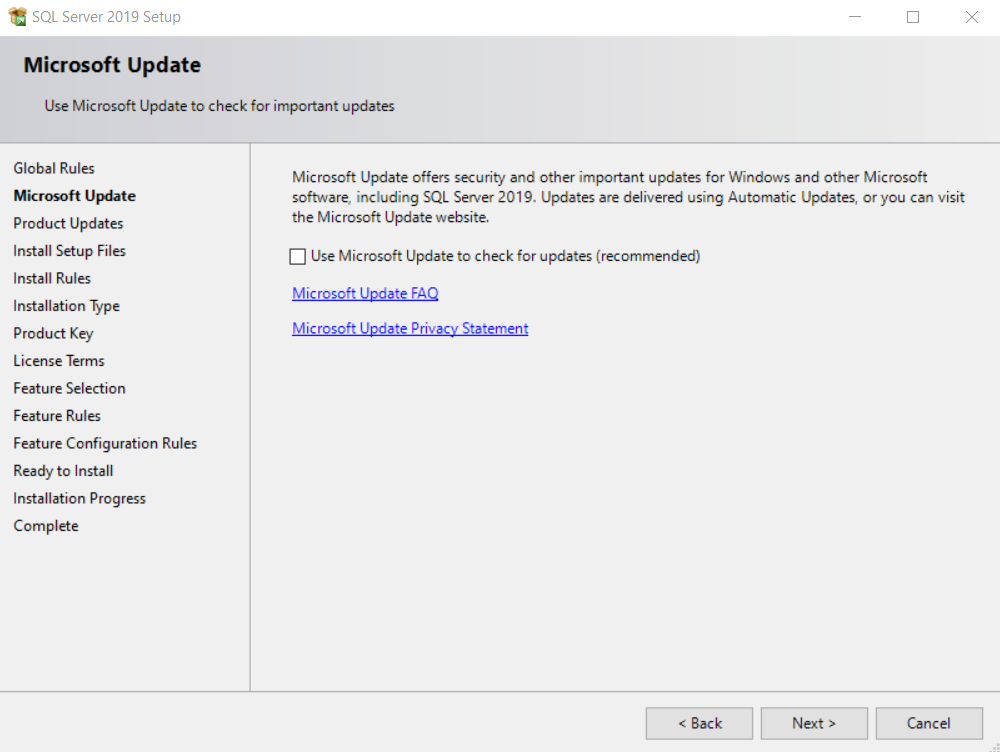
Choose Installation:

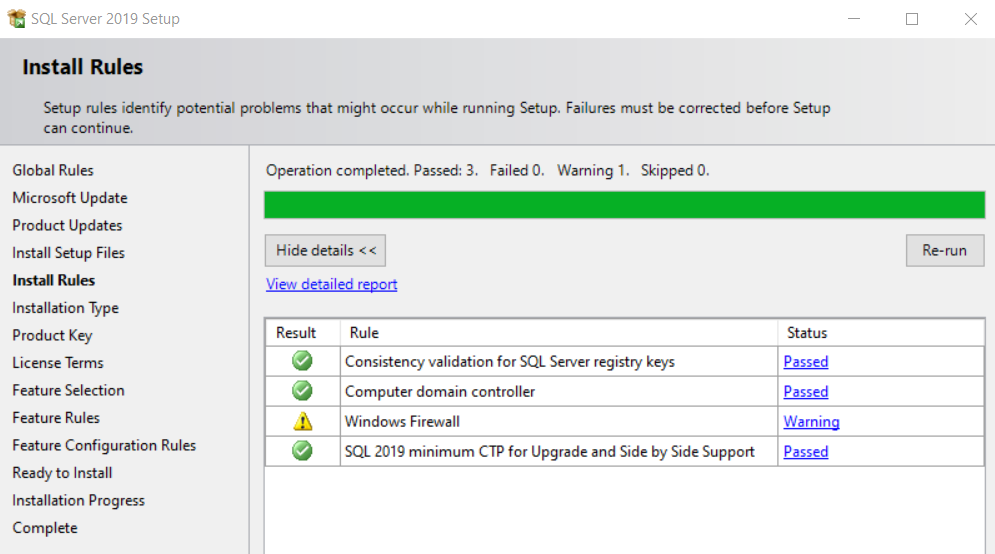


Click:

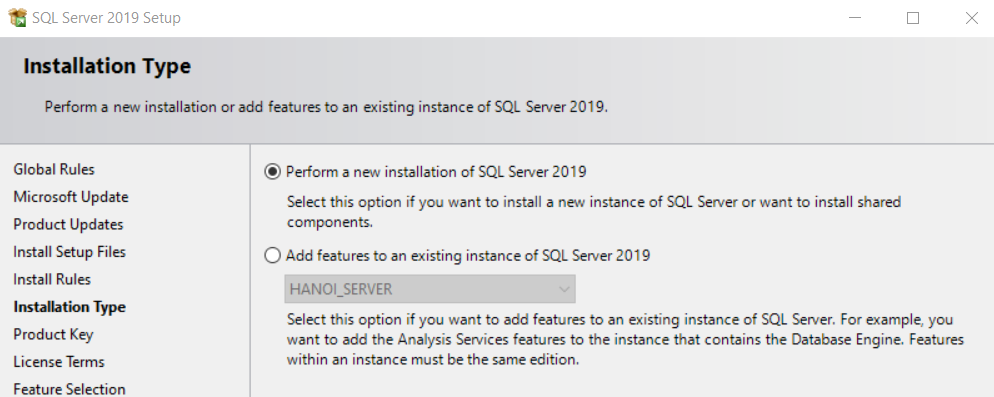


Click Next:

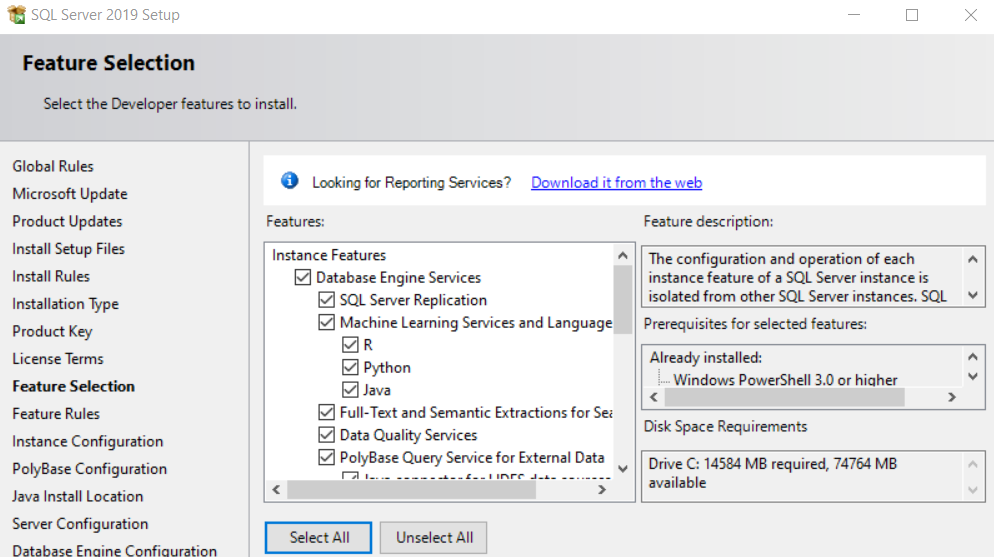


Here appears an error message about Windows Firewall, but without compromising, continue to press next:

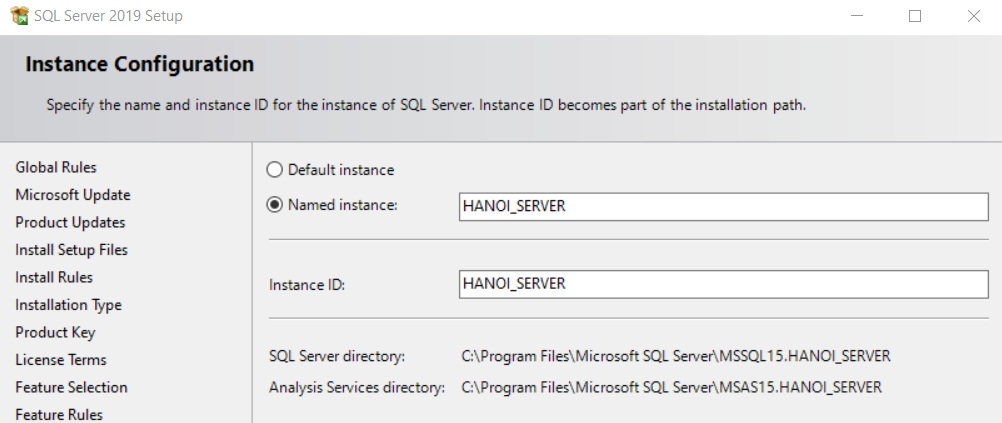
Select "Perform a new ..." And then continue next:



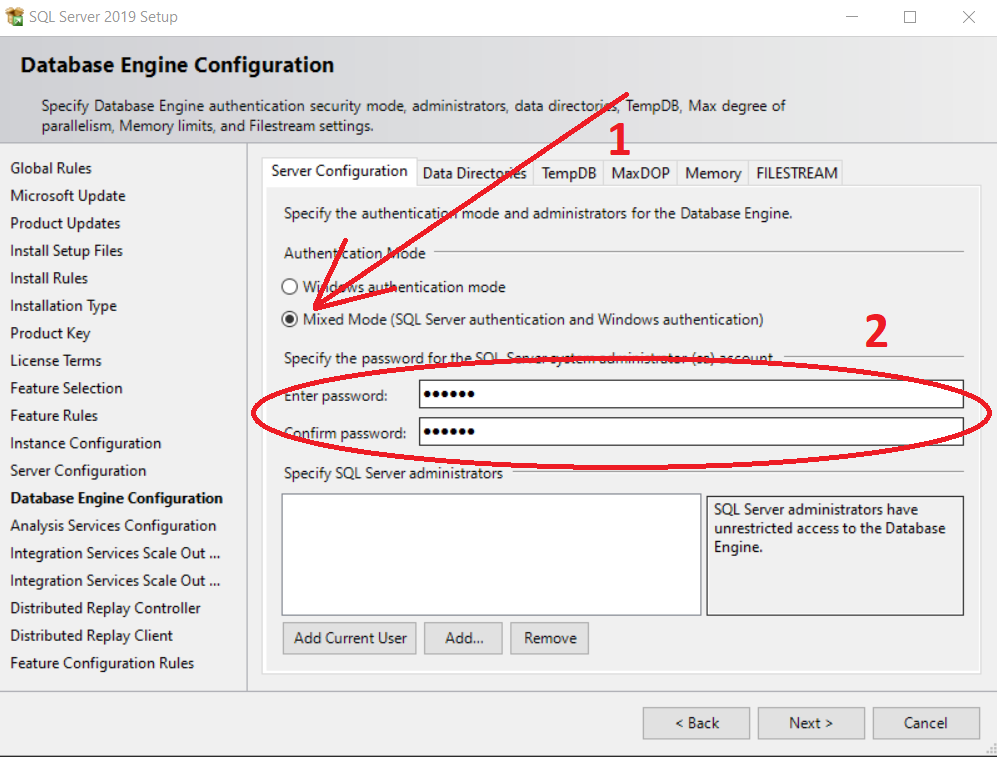
Select Select All (or you can choose according to your needs):



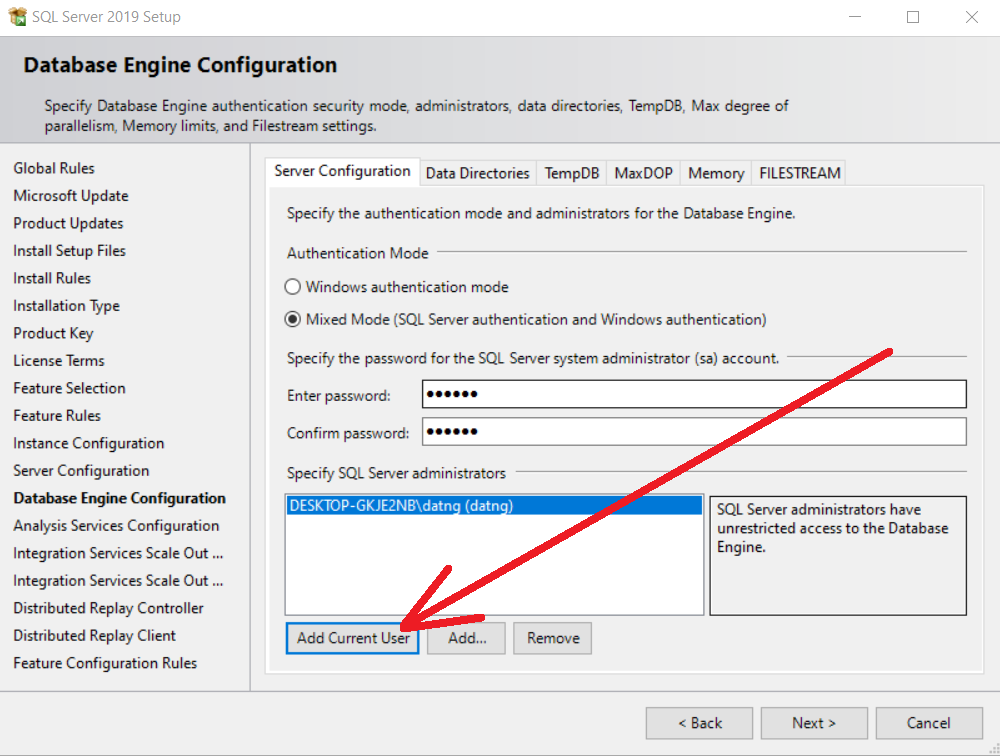
Enter a name in the Named Instance section:



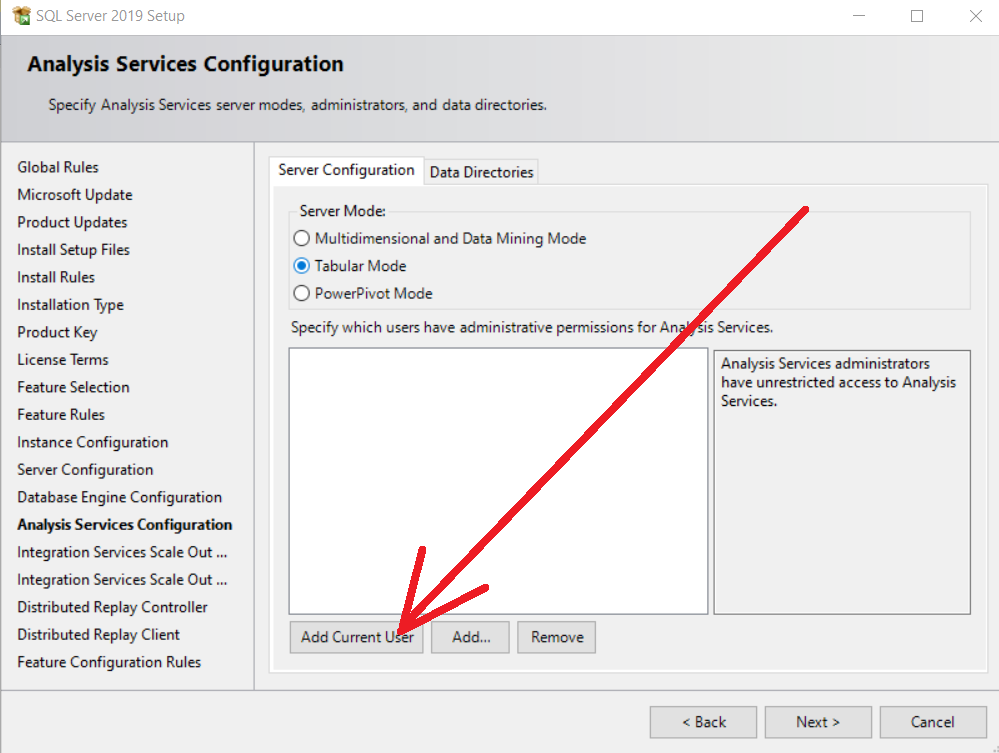
Select Mixed Mode… and enter Password:



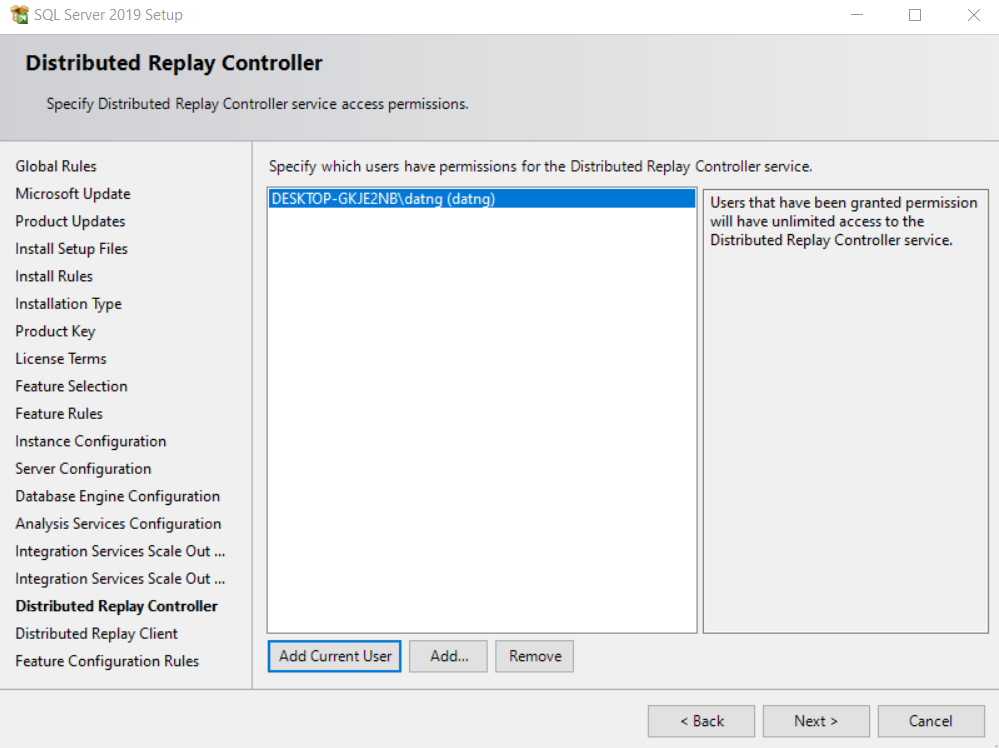
Select Add Current User and then Next:



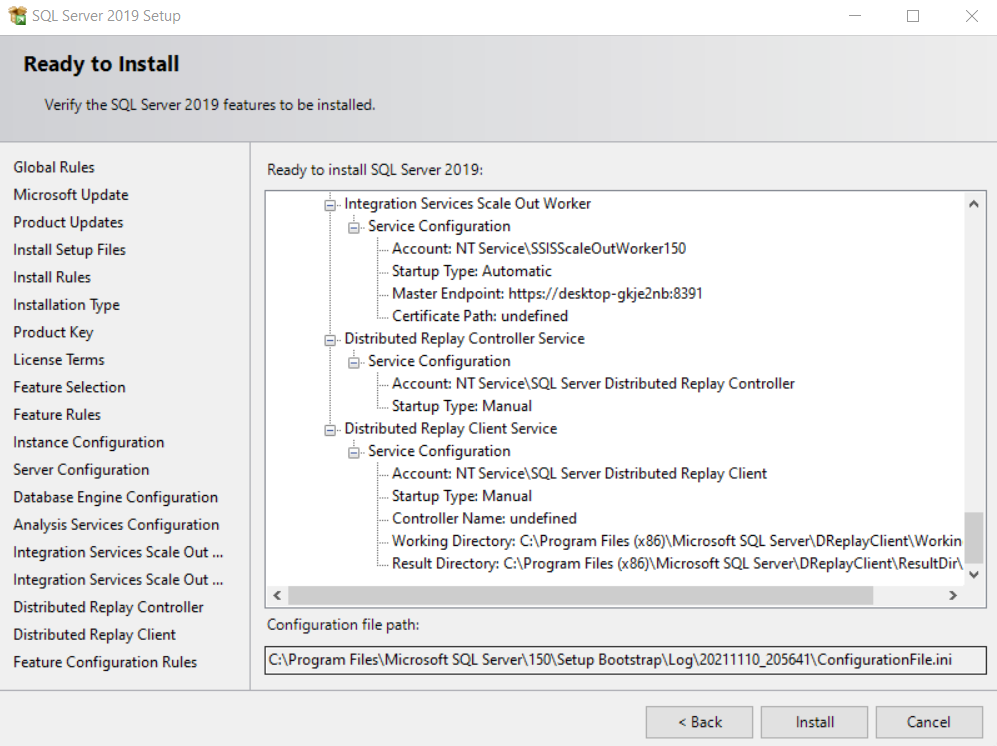
Continue to select Add Current User and click Next:



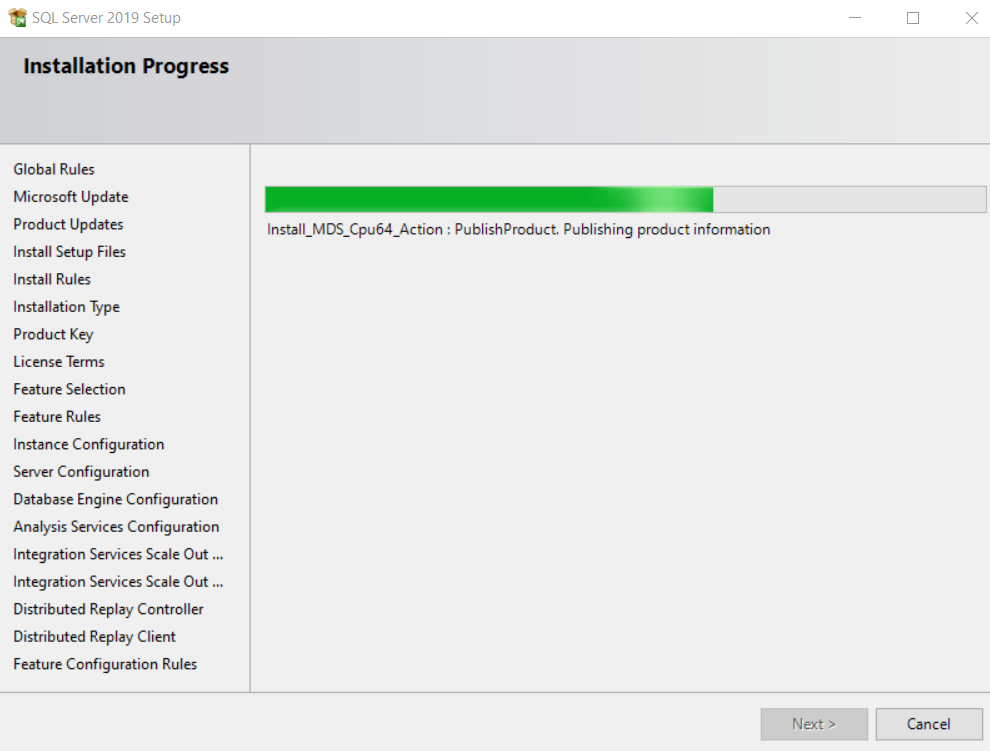
Select Add Current User and click Next:



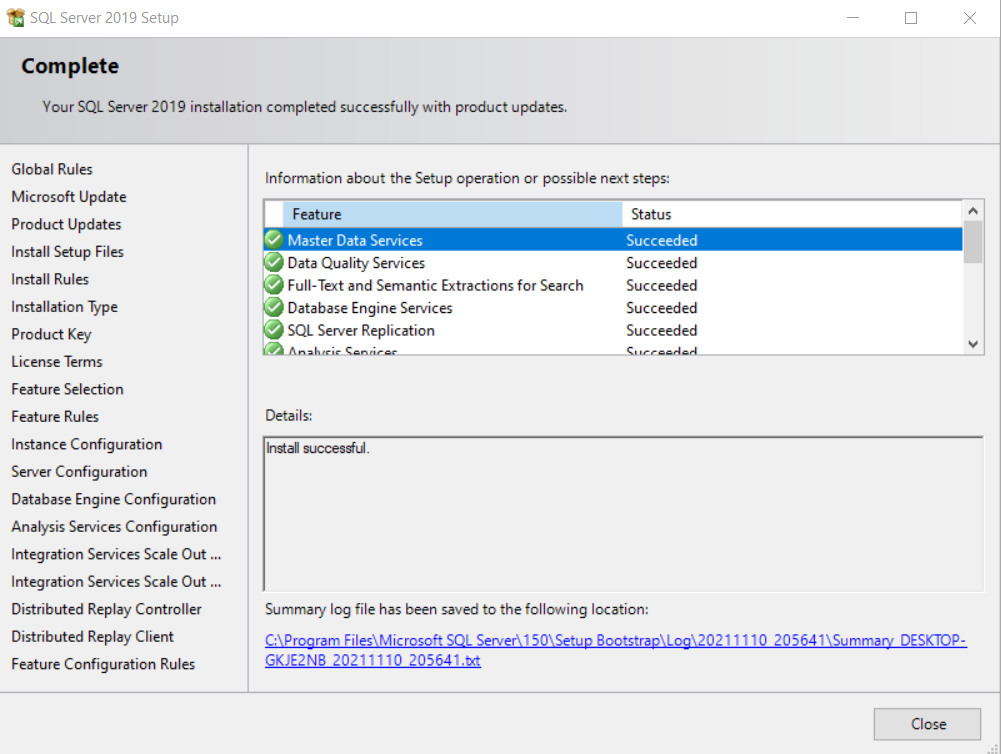
Installation confirmation 🡪 Select Install to proceed with the installation:



During the self-installation process:



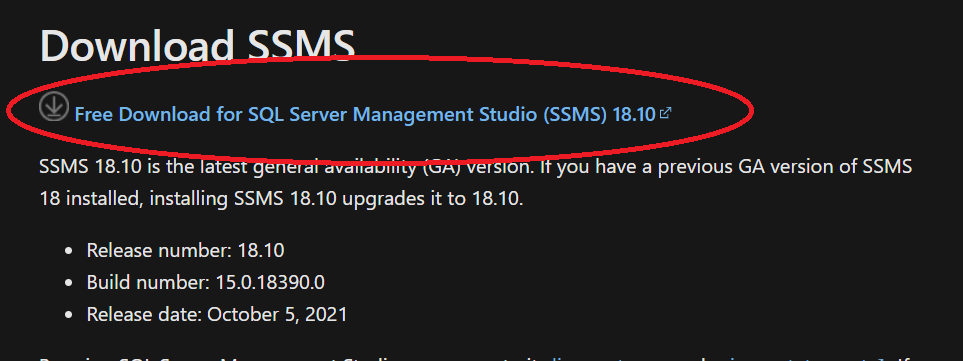
Installation is complete, click Close to complete:



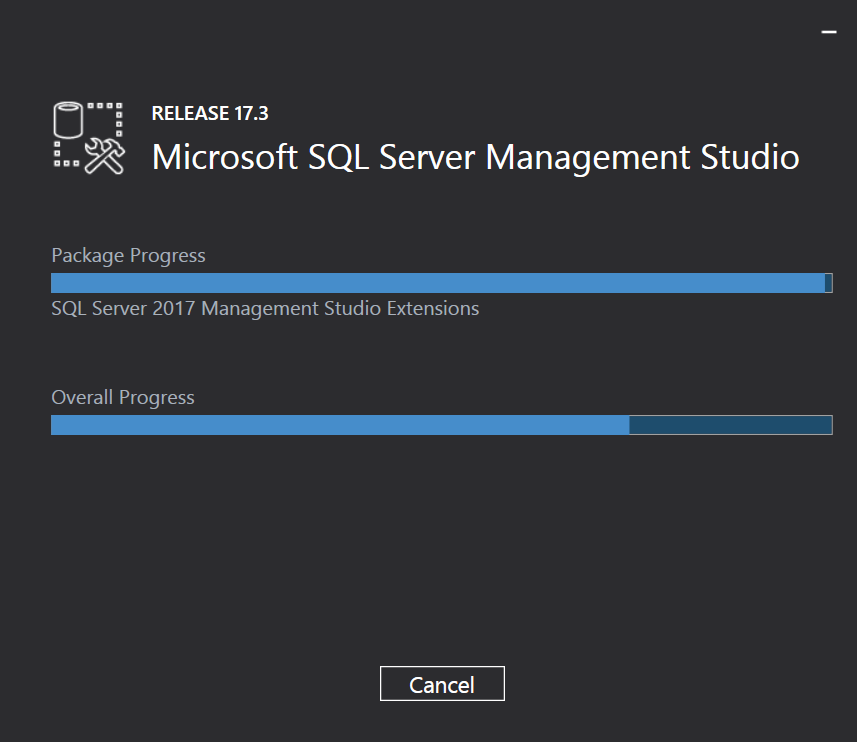
## **Setting SMSS**

SSMS = SQL Server Management Studio (installed on all machines)

Download link: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>



\* Note: some versions are available, after installing SQL server, SSMS installation will appear. If not, you can download it manually from the link above

The SSMS installation process is simple, just press Next to the program to install automatically

## **Setting Radmin VPN**

Installed on all machines

Use to create a virtual private network (VPN, which establishes a secure connection between computers over the Internet just like computers connect to each other on a LAN

Download here: <https://www.radmin-vpn.com/>

Automatic installation doesn't require a lot of config

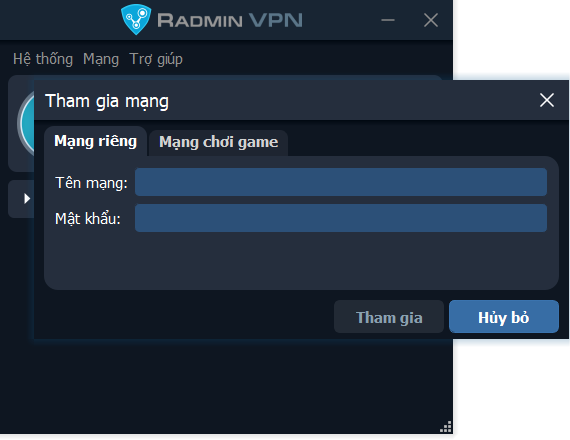
Choose Create Network:



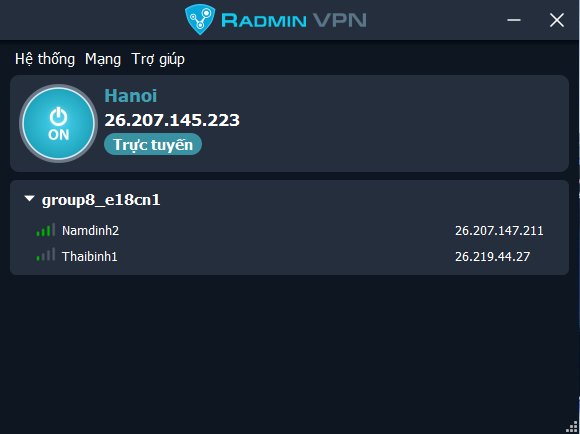
Fill in Name, Password:



(Theme and Language can be adjusted appropriately)

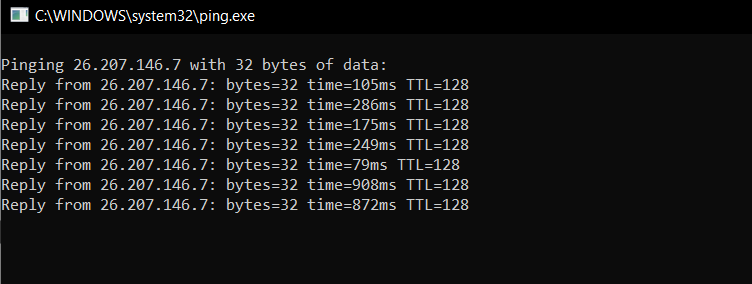
Workstations select Join Network:

Then enter the network name and password like in the installed server to join the Network:



Fill in information such as network name and password. The workstations will connect to each other and connect to the server using this VPN.

Check the connection by right-clicking on the member and clicking ping, the output as shown below means the machines have connected successfully.

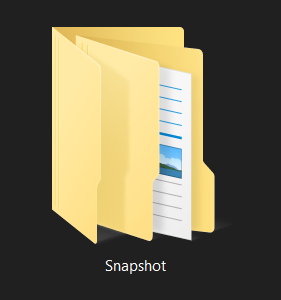


If this is the connection is successful.

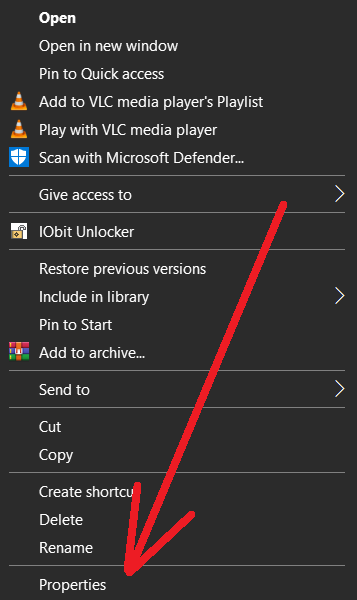
# **Configuration**

## **Create Shared Folder**

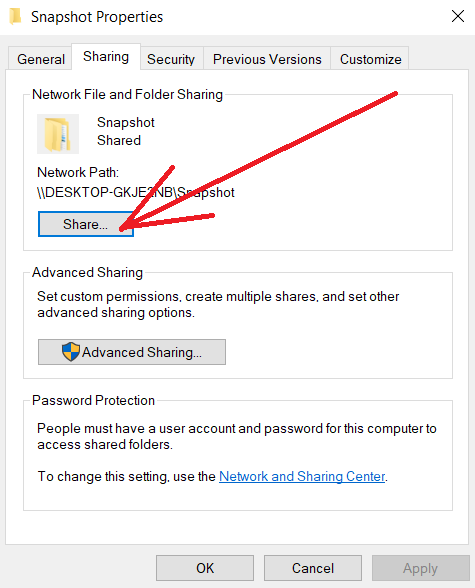
Create folder Snapshot:



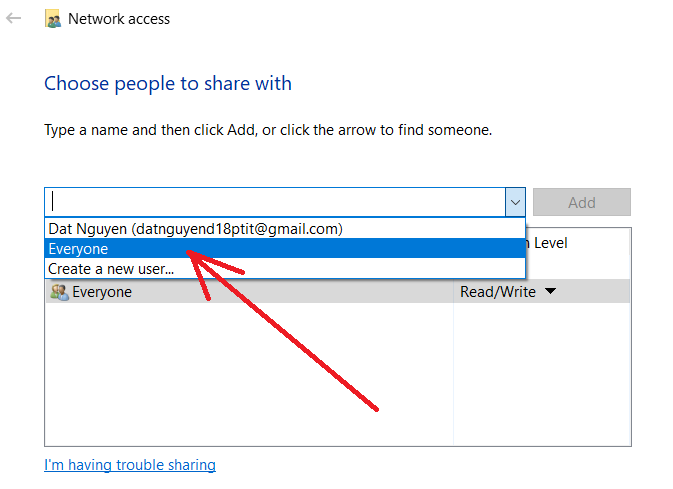
Right mouse 🡪 Properties:



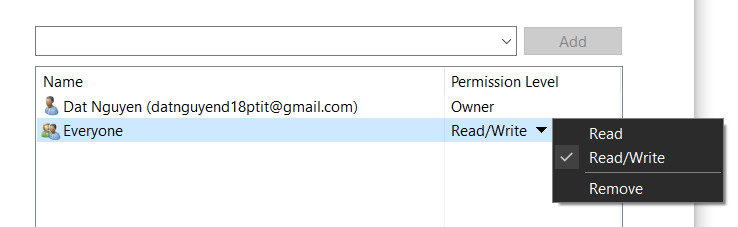
Click Sharing and click “Share..”



Click Everyone:

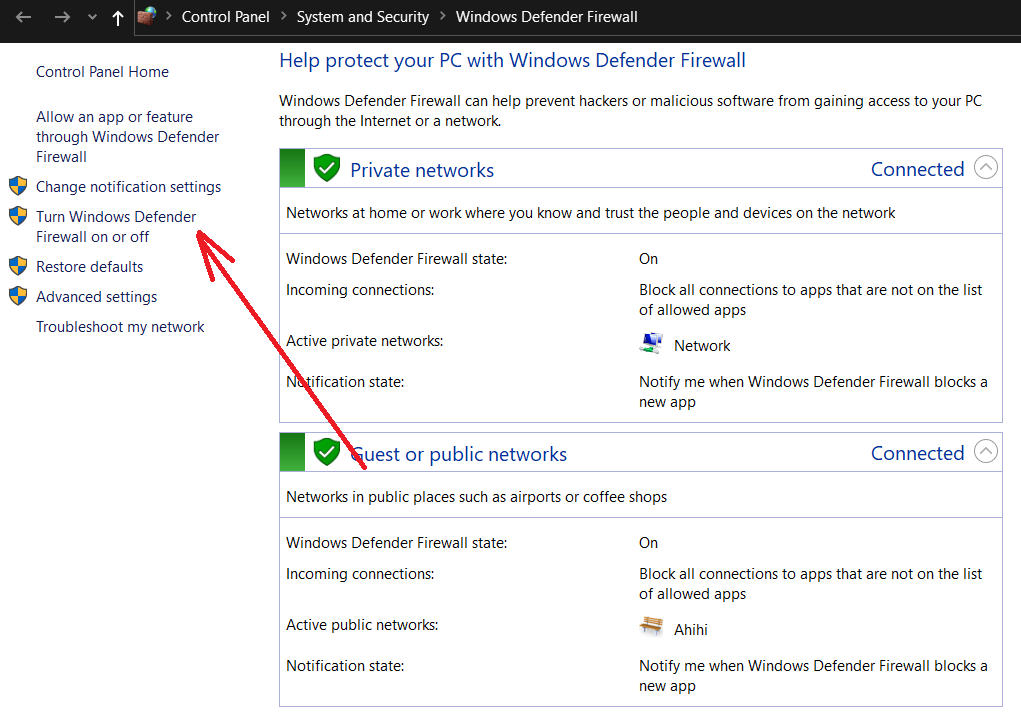


Click Read/Write:

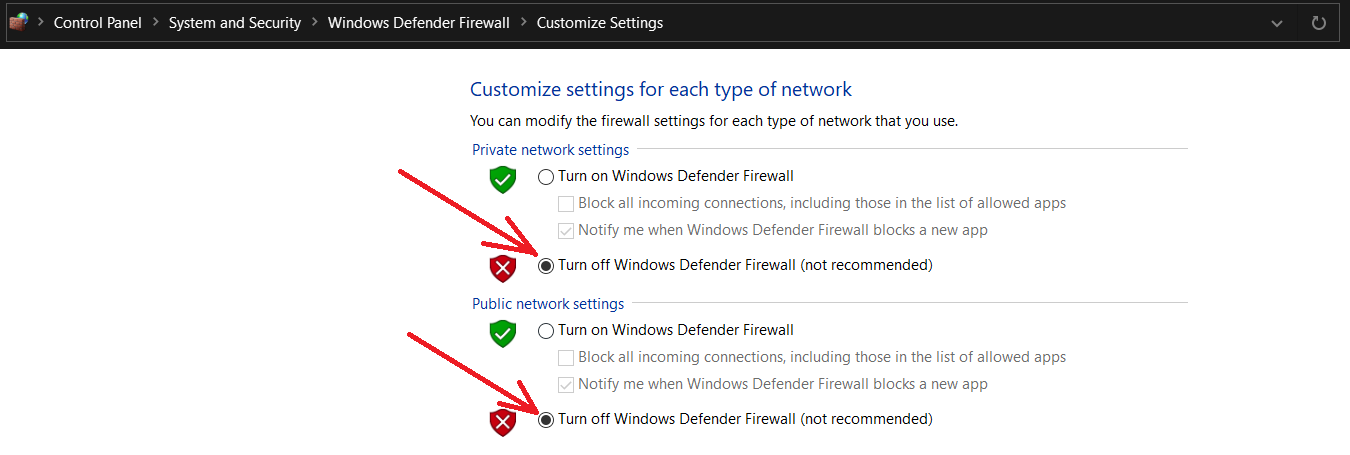


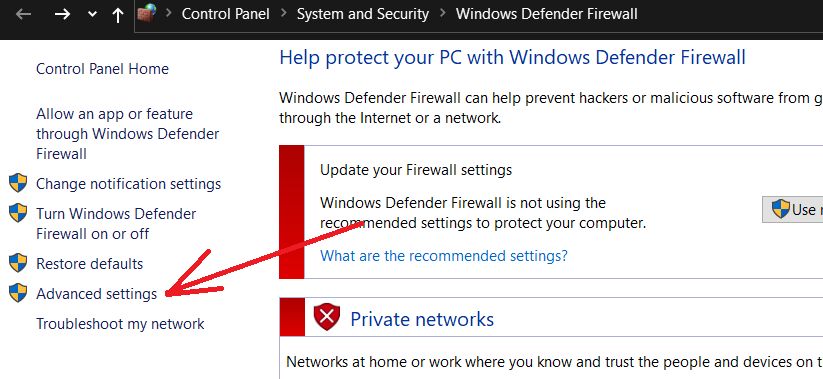
## **Setting Firewall**

Control Panel\System and Security\Windows Firewall:

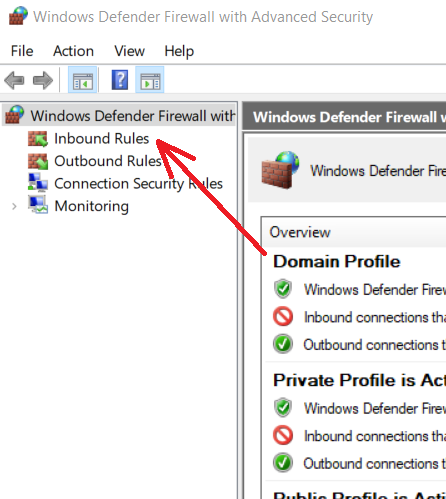


Click “Turn off …” firewall (optional):

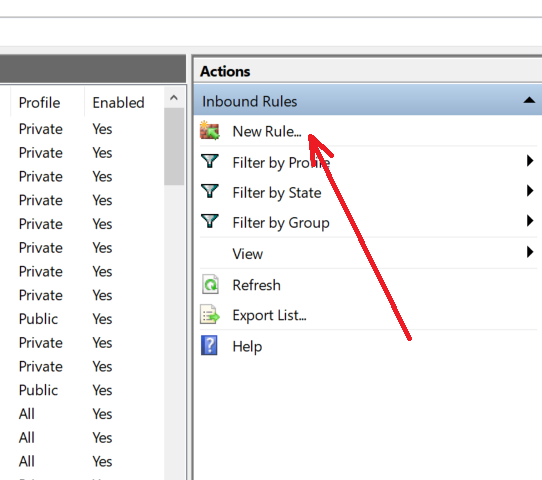


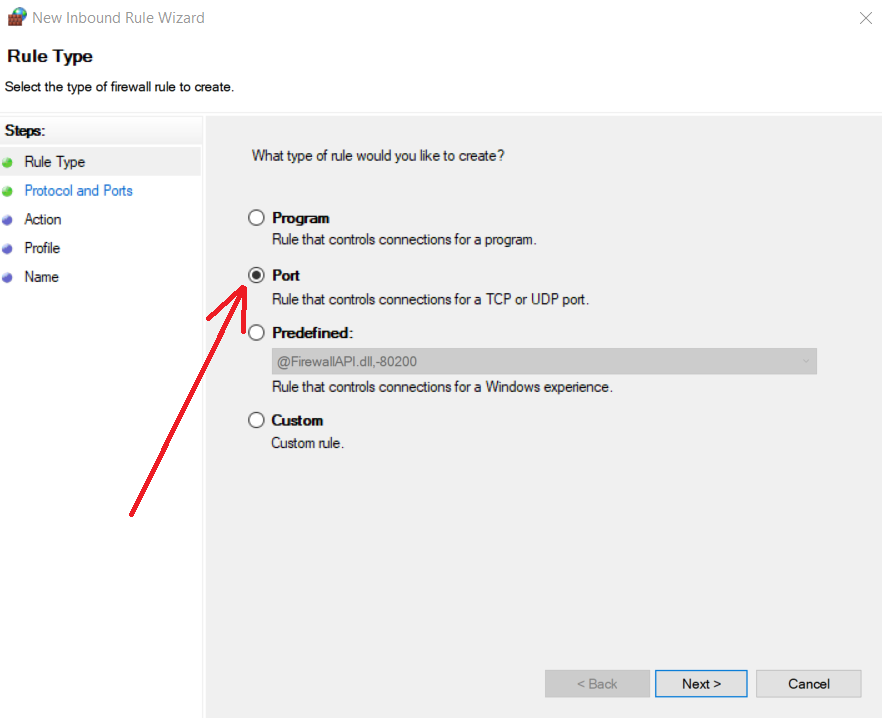


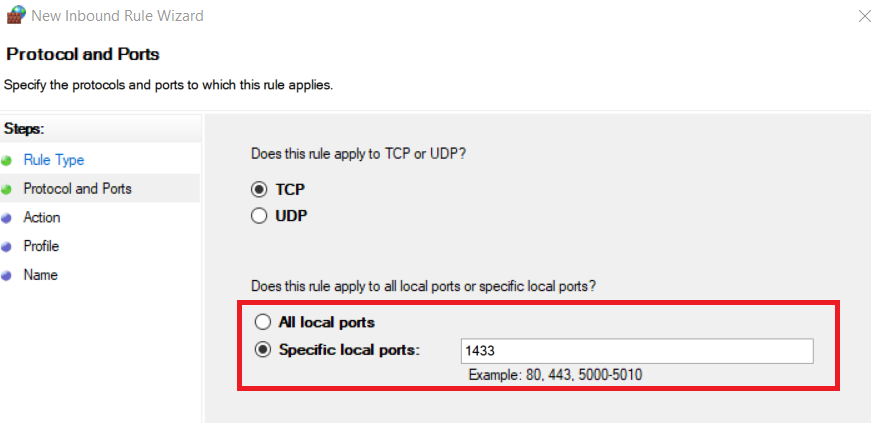
Click Inbound Rules:



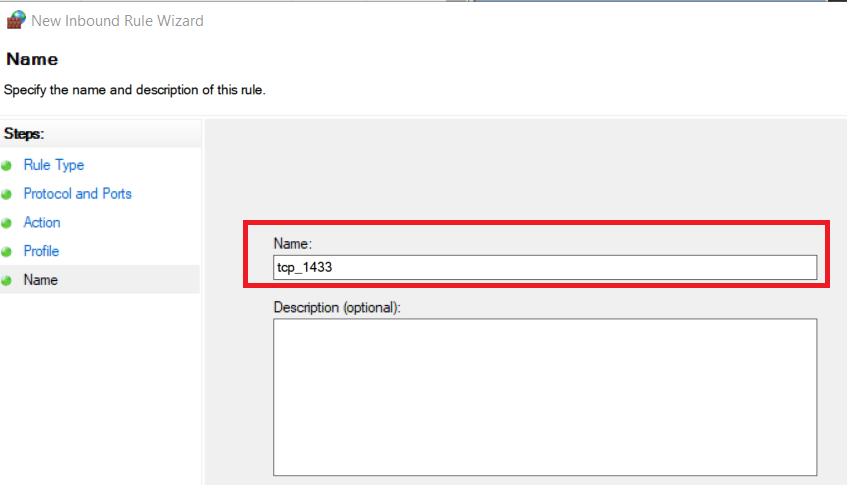
Click New Rule … :







Click next untill:

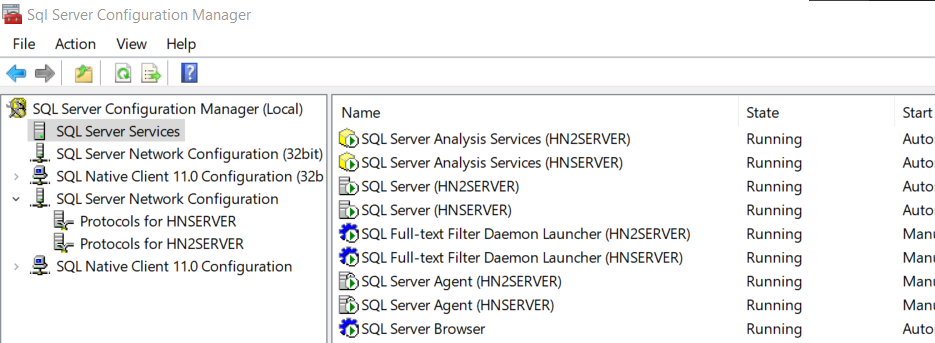


## **Configuration IP**

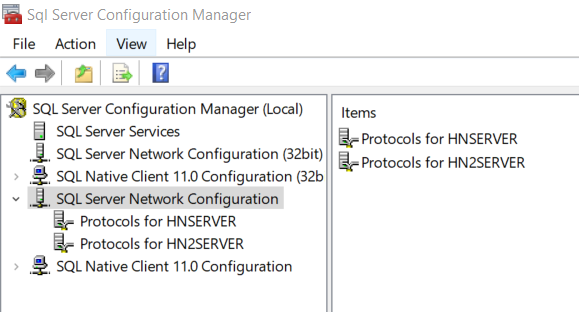
Configuration on both servers and workstations

Open SQL Server 2019 Configuration Manager:

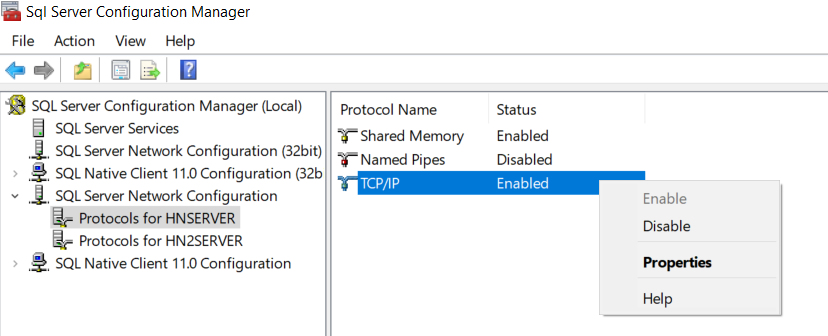




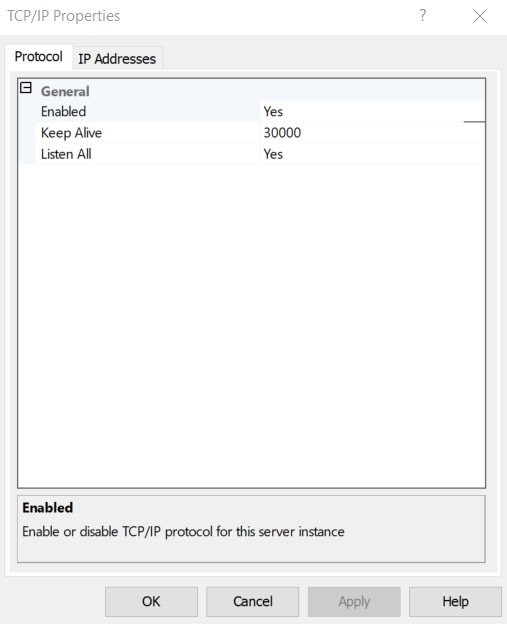
Click SQL Server Network Configuration:



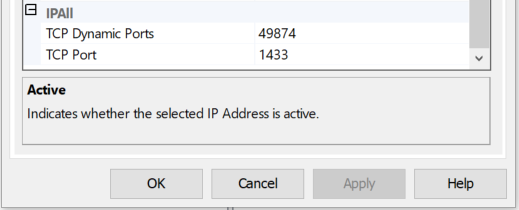
Set TCP/IP to Enabled:

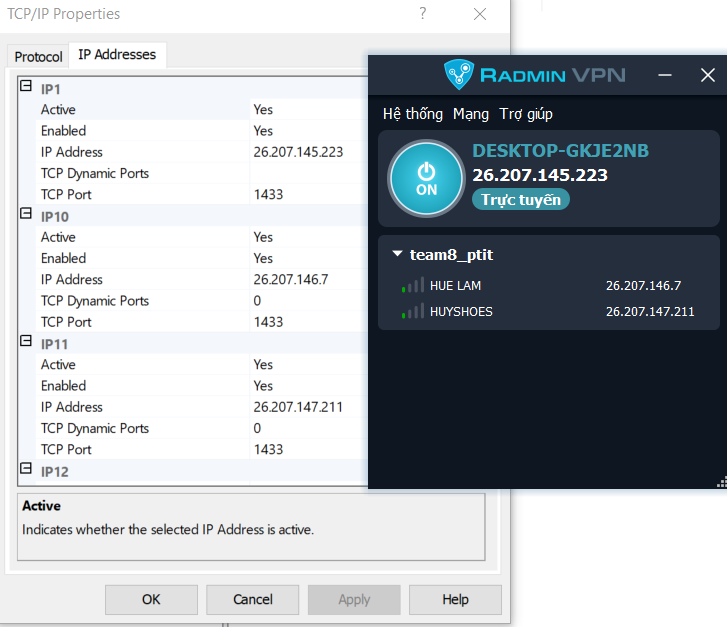


Open TCP/IP:

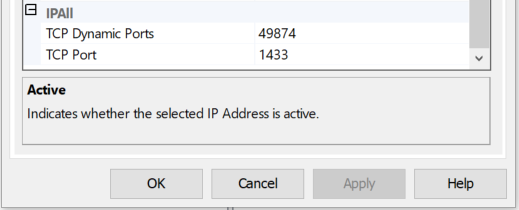


Select IP Address:



And configured for each IP item (for both servers and workstations)

Drag to the bottom and adjust the IPALL section to 1433:

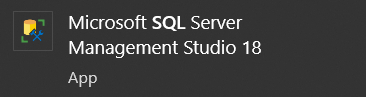


Noted: If setting in 1 computer, must use diffirent port (1434, 1435,…)

# **Database settings and fragmentation**

1. **Install a global database**

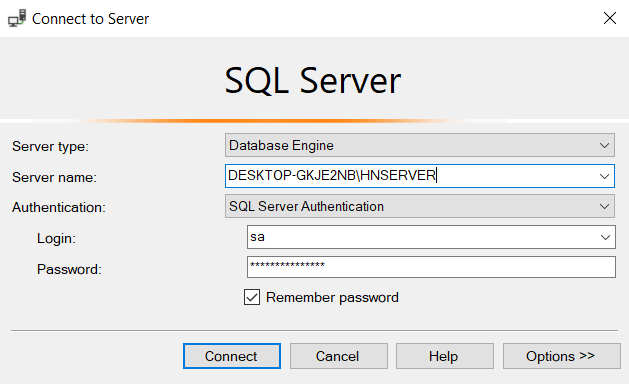
Open SQL Server Management Studio 18

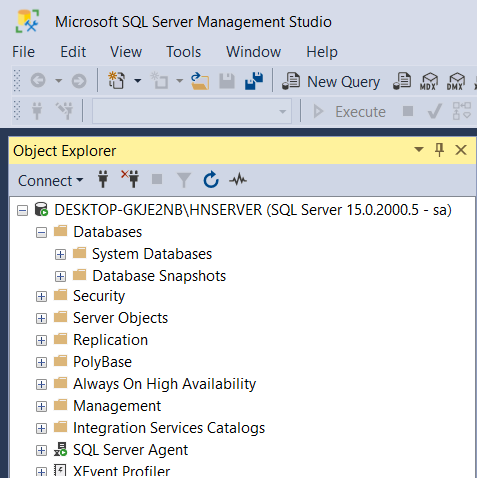


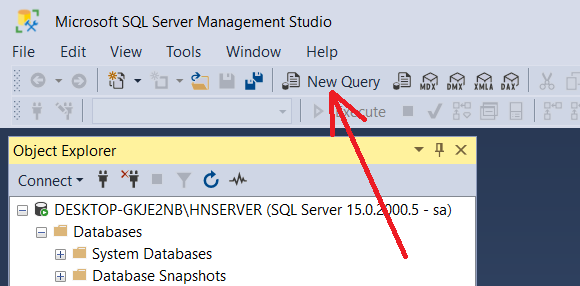
Sign in with the following account:

**Username: sa**

**Password: 12345**

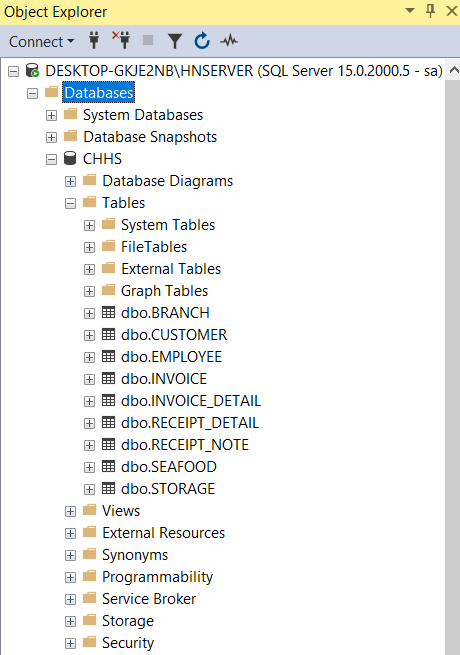






|  |
| --- |
| CREATE DATABASE CHHS  GO  ----------------------------  ----------------------------  USE CHHS  GO  -----------------------------------  ---BRANCH\_TABLE  CREATE TABLE BRANCH(  ID\_BRA nchar(25) PRIMARY KEY,  NAME\_BRA nvarchar(255) UNIQUE,  ADDR nvarchar(255),  NUM nchar(25)  )  ---CUSTOMER\_TABLE  CREATE TABLE CUSTOMER(  ID\_CUS nchar(25) PRIMARY KEY not null,  FULLNAME nvarchar(255),  ADDR nvarchar(255),  NUM nchar(25),  ID\_BRA nchar(25) FOREIGN KEY (ID\_BRA) REFERENCES BRANCH(ID\_BRA) ON UPDATE CASCADE  )  ---STORAGE\_TABLE  CREATE TABLE STORAGE(  ID\_STO nchar(25) PRIMARY KEY,  NAME\_STO nvarchar(255) UNIQUE,  ADDR nvarchar(255),  ID\_BRA nchar(25) FOREIGN KEY (ID\_BRA) REFERENCES BRANCH(ID\_BRA) ON UPDATE CASCADE  )  ---EMPLOYEE\_TABLE  CREATE TABLE EMPLOYEE(  ID\_EMP char(25) PRIMARY KEY not null,  FULLNAME nvarchar(255),  AGE int,  ADDR nvarchar(255),  SAL float CHECK(SAL > 5000000.0),  ID\_BRA nchar(25) FOREIGN KEY (ID\_BRA) REFERENCES BRANCH(ID\_BRA) ON UPDATE CASCADE  )  ---GOODS\_TABLE  CREATE TABLE SEAFOOD(  ID\_SEA char(25) PRIMARY KEY not null,  NAME\_SEA nvarchar(255),  PRICE float,  SUPPLIER nvarchar(255),  INSTOCK float CHECK(INSTOCK > 0.0)  )  ---INVOICE\_TABLE  CREATE TABLE INVOICE(  ID\_INV int PRIMARY KEY not null,  TIME\_DATE smalldatetime DEFAULT(GETDATE()),  TOTALPRICE float,  ID\_CUS nchar(25) FOREIGN KEY (ID\_CUS) REFERENCES CUSTOMER(ID\_CUS),  ID\_EMP char(25) FOREIGN KEY (ID\_EMP) REFERENCES EMPLOYEE(ID\_EMP),  ID\_STO nchar(25) FOREIGN KEY (ID\_STO) REFERENCES STORAGE(ID\_STO) ON UPDATE CASCADE  )  ---INVOICE\_DETAIL\_TABLE  CREATE TABLE INVOICE\_DETAIL(  ID\_INV int FOREIGN KEY (ID\_INV) REFERENCES INVOICE(ID\_INV) ON UPDATE CASCADE,  ID\_SEA char(25) FOREIGN KEY (ID\_SEA) REFERENCES SEAFOOD(ID\_SEA) ON UPDATE CASCADE,  AMOUNT float CHECK (AMOUNT > 0.0)  )  ---RECEIPT\_NOTE\_TABLE  CREATE TABLE RECEIPT\_NOTE(  ID\_REC int PRIMARY KEY,  DATE\_TIME date DEFAULT(GETDATE()),  SOURCE nvarchar(255),  TOTALPRICE float,  ID\_EMP char(25) FOREIGN KEY (ID\_EMP) REFERENCES EMPLOYEE(ID\_EMP),  ID\_STO nchar(25) FOREIGN KEY (ID\_STO) REFERENCES STORAGE(ID\_STO) ON UPDATE CASCADE  )  ---RECEIPT\_DETAIL\_TABLE  CREATE TABLE RECEIPT\_DETAIL(  ID\_REC int FOREIGN KEY (ID\_REC) REFERENCES RECEIPT\_NOTE(ID\_REC) ON UPDATE CASCADE,  ID\_SEA char(25) FOREIGN KEY (ID\_SEA) REFERENCES SEAFOOD(ID\_SEA) ON UPDATE CASCADE,  AMOUNT float CHECK (AMOUNT > 0.0),  PRICE float  ) |

After running the Query we get the database:

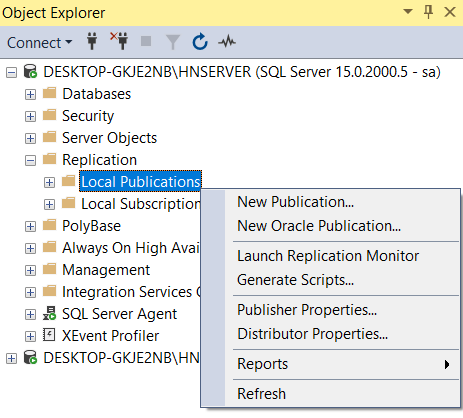


**2. Data fragmentation (Create Publication Database)**

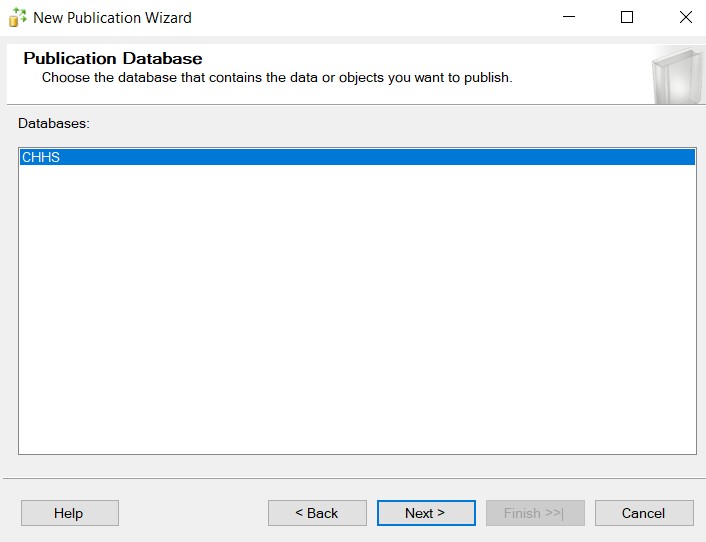
**2.1. Horizontal fragmentation**

Horizontal database fragmentation by branches Thai Binh (“BRA1”) and Nam Dinh (“BRA2”)

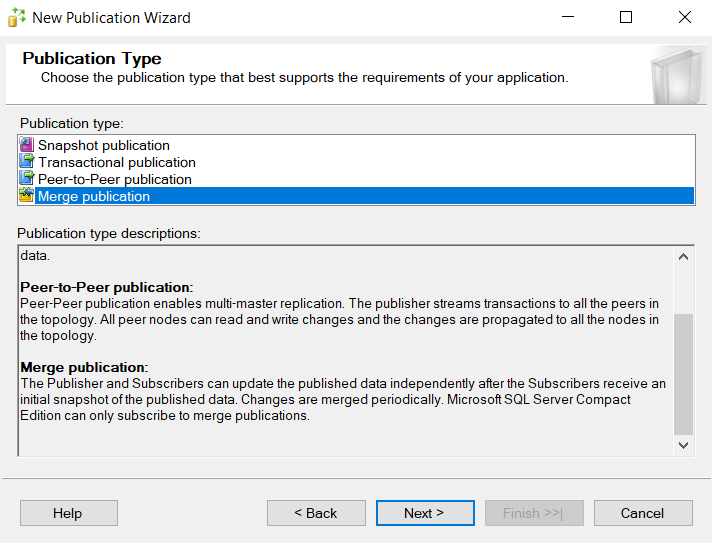
- Go to Replication, right-click Local Publications and select New Publication



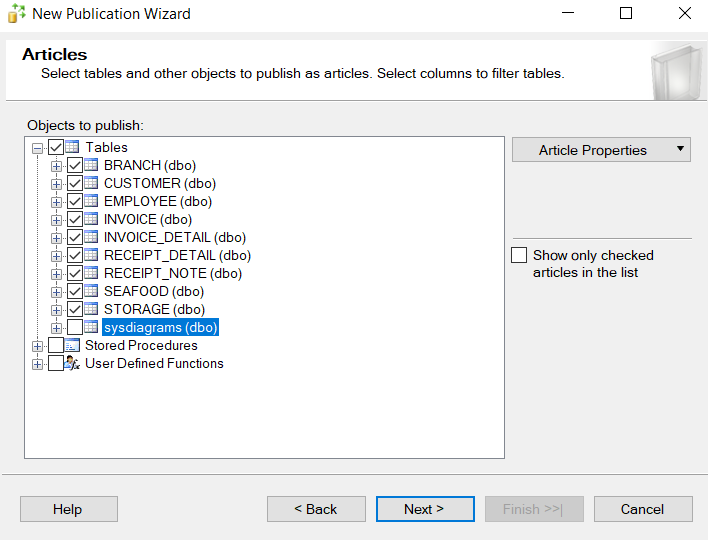
* Choose database



* Click Next and choose Merge publication

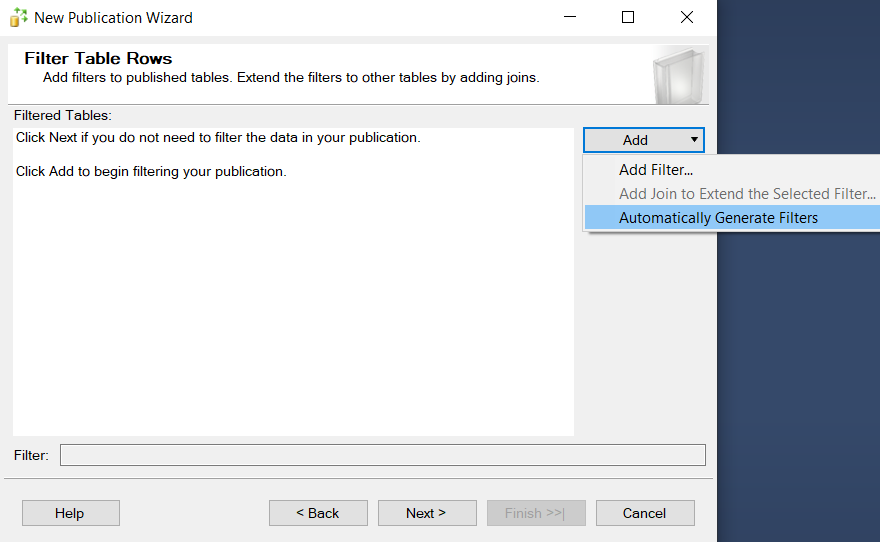


Click Next until this display, choose Table :



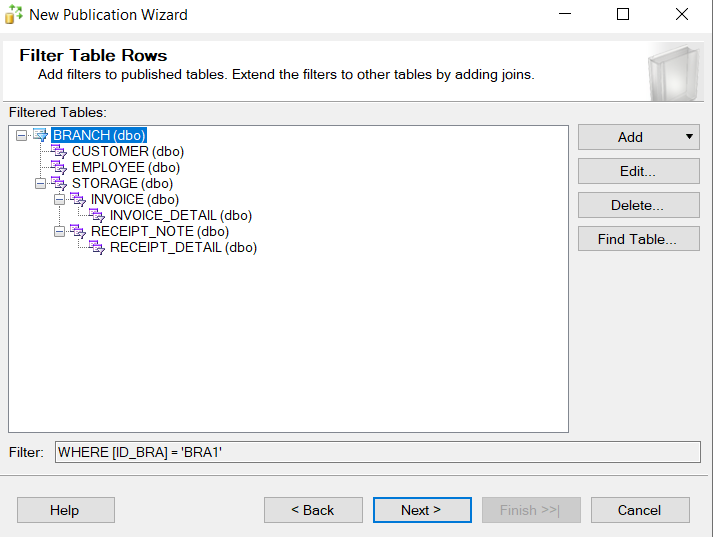
Note : dont chose sysdiagram (dbo)

* Choose Next -> Click Automatically Generate Filter



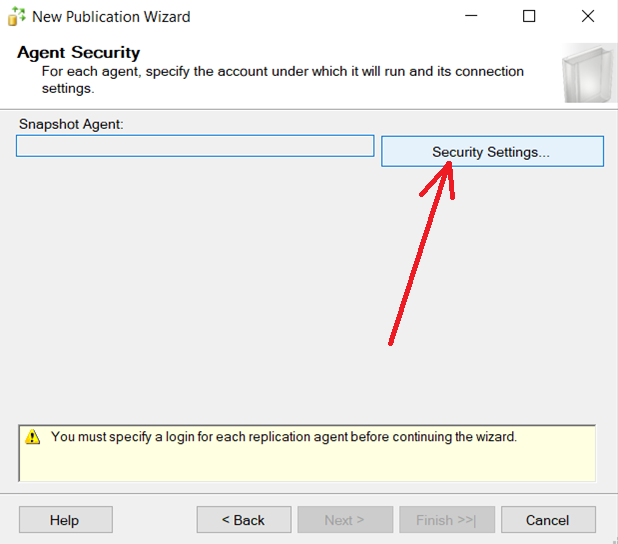
Choose like this :



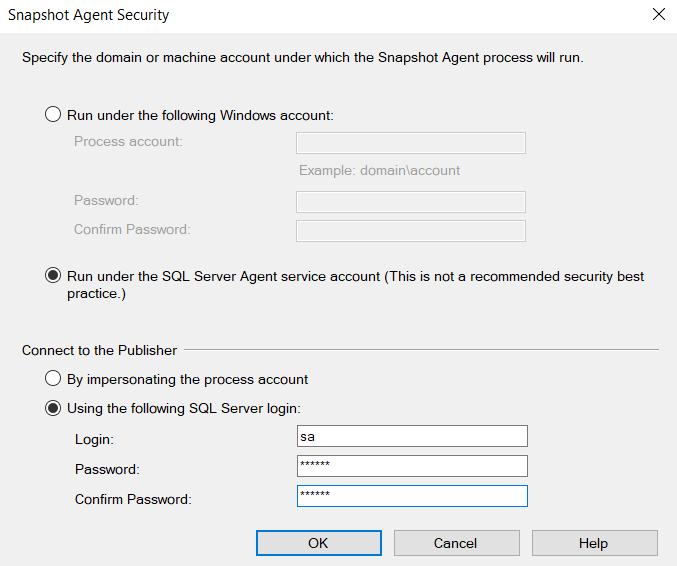




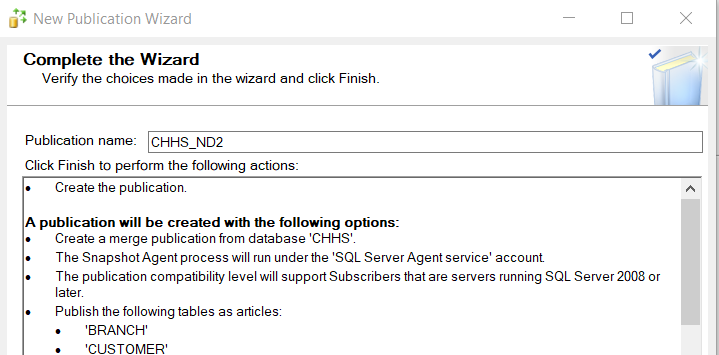
* Click nexy until this Secủity Setting :



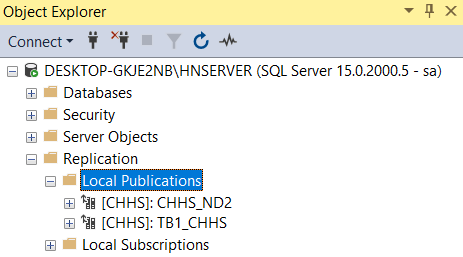
* Choose this :



* Name the publication:



* Click Finish and wait system get start:

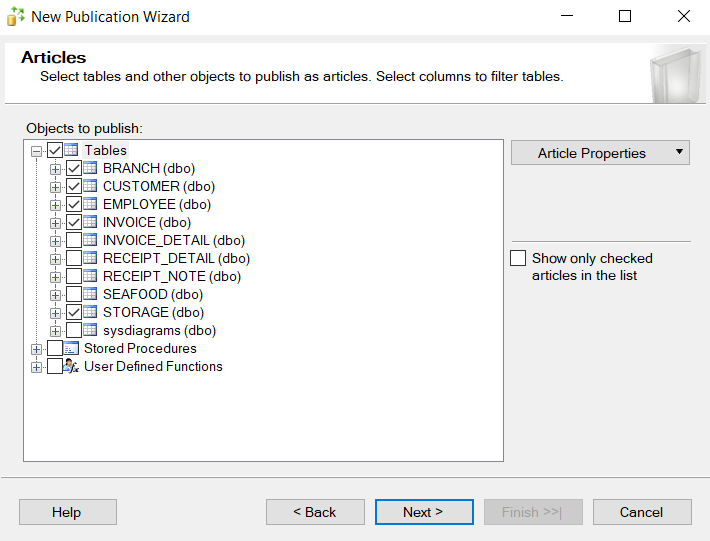


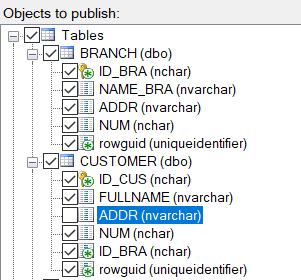
### ***Vertical fragmentation***

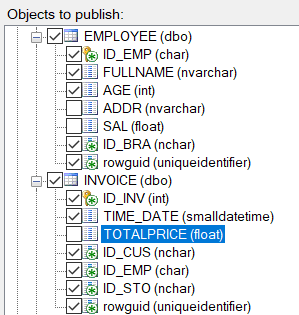
Vertical fragmentation of the database to manage customers and employee of each branch

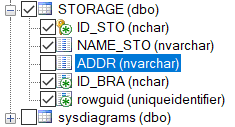
Do the same steps as horzontal fragmentation. Only the step is different :

Select only the Tables and columns that will perform fragmentation:





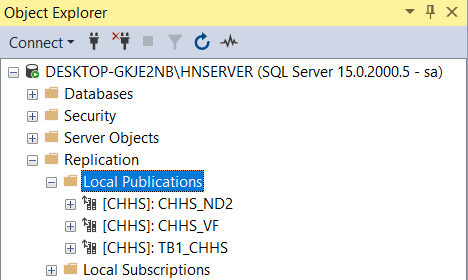




Click next:



The rest of the stép are similar

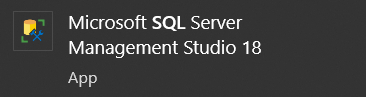


# **Connect, Decentralize – ensure transparency**

## **Configure the “sa” account**

The server has read and edit rights to the database on the stations. So need to configure sa account on all stations:

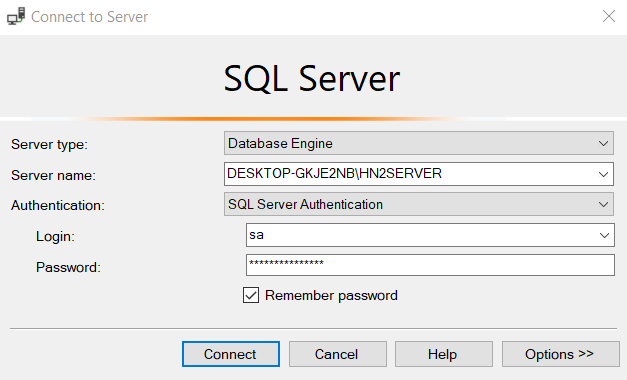
On the stations, open SQL Server Management Studio 18:



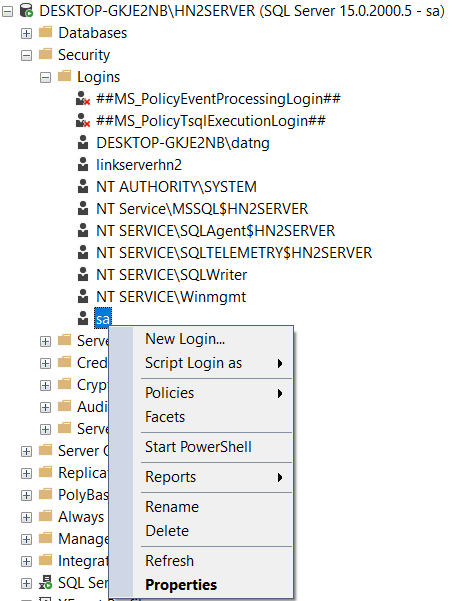
sa:

***Username: sa***

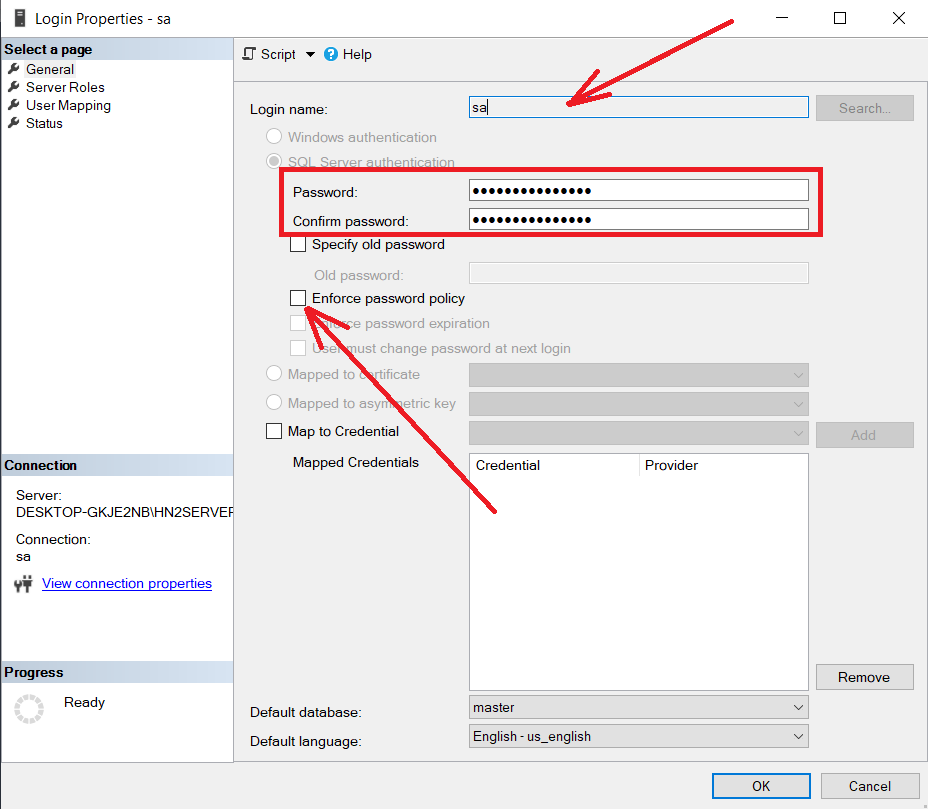
***Password: 12345***



Open Security 🡪 Logins 🡪 Right click to “sa” 🡪 Chose Properties:



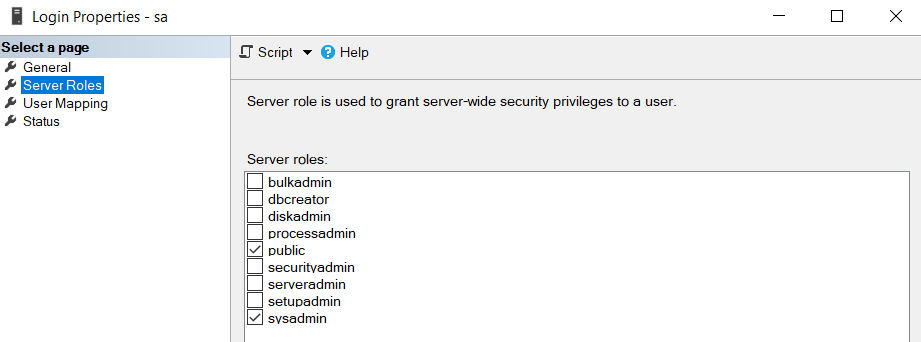
Configure like this :



Login name: sa

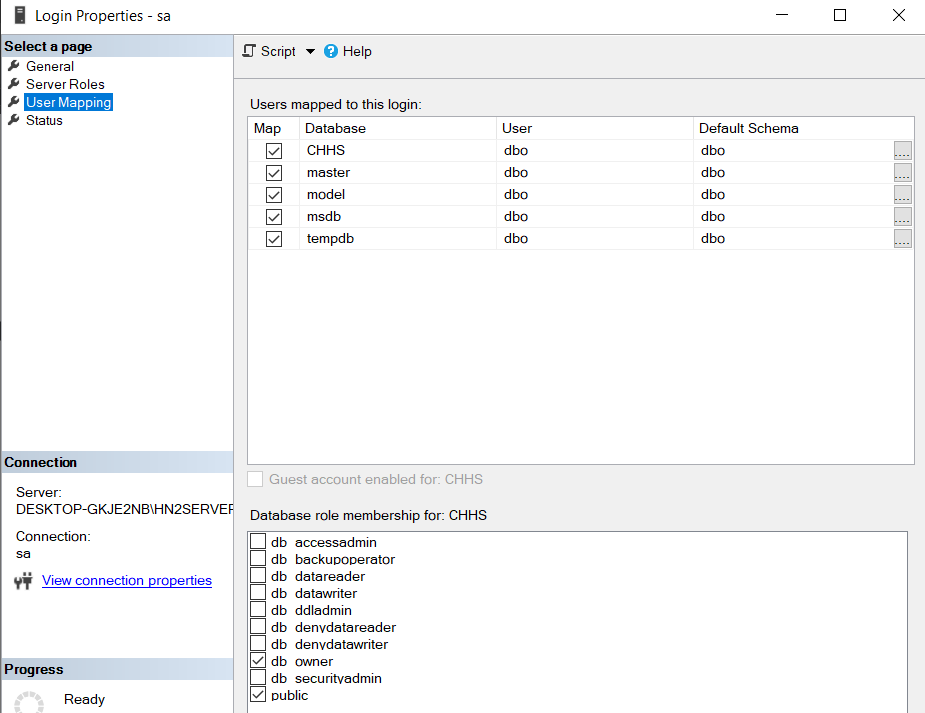
Password: 123456

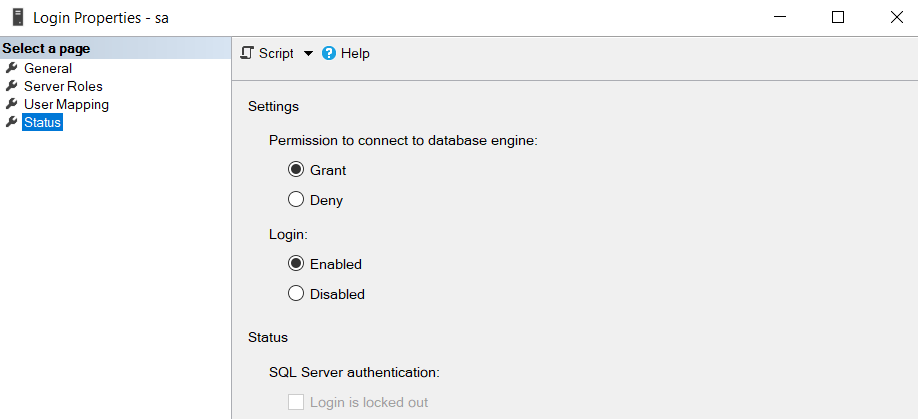
Unchecked “Enforce password policy”



In Database role membership for:

Choose “db owner” (Full permission)

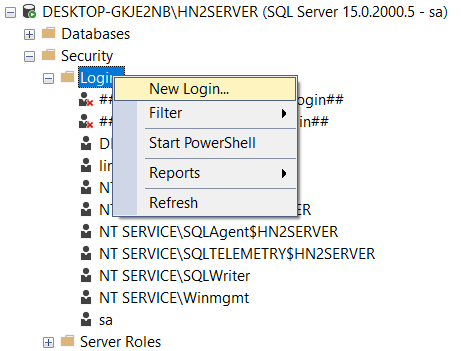




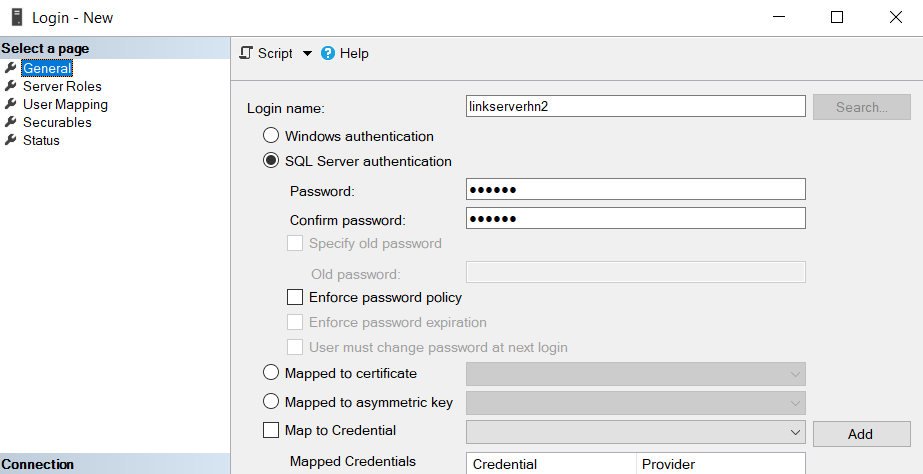
## **Create a read-only account**

Between station that need to connect with each other and only have the right to read data on other stations. So need to create a read-only account

Security - > Login -> New Login



Config as follow:

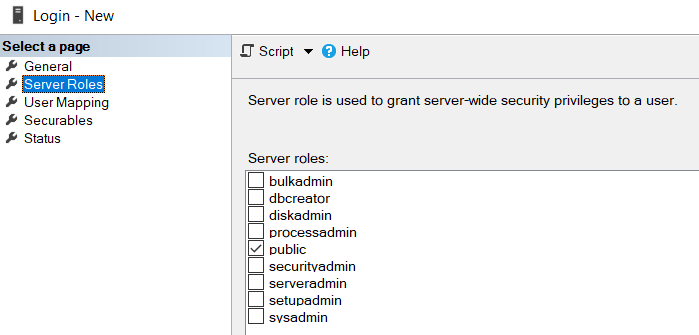


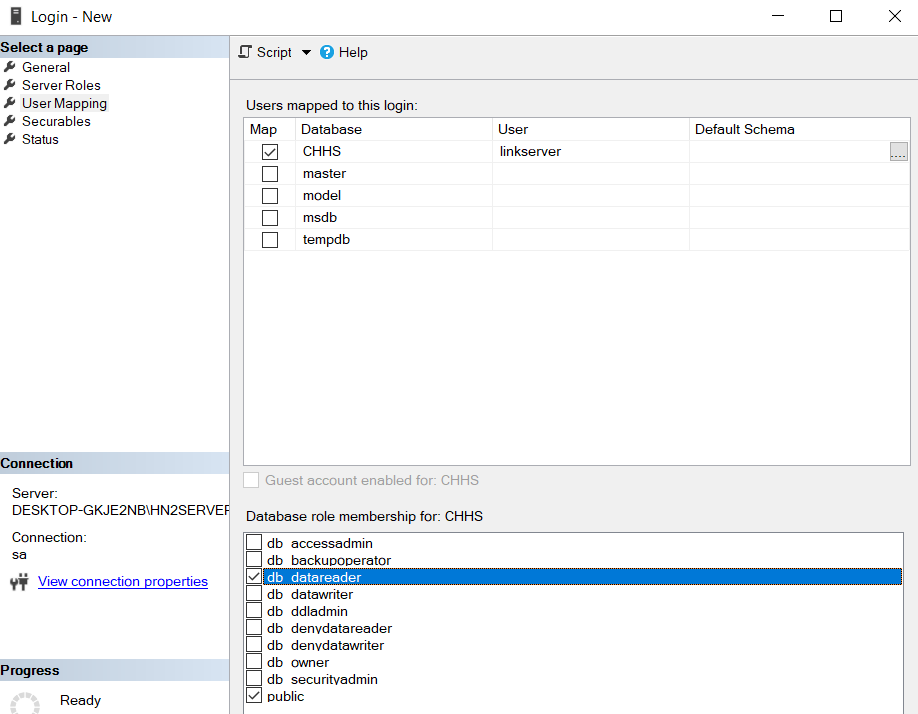
With login name correspondung to each station:

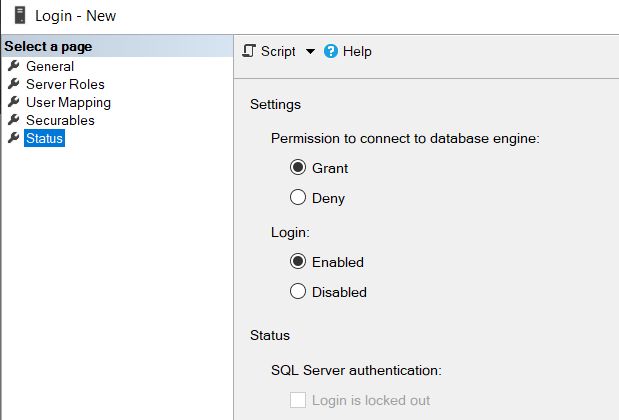
**Thai Binh:** linkservertb1

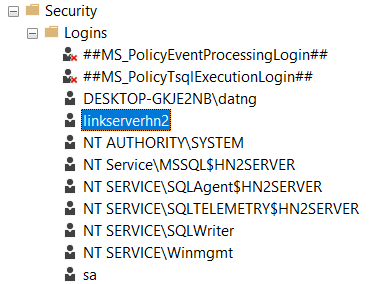
**Nam Dinh:** linkservernd2

**Ha Noi 2:** linkserverhn2



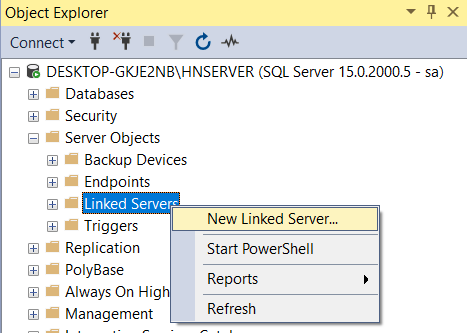




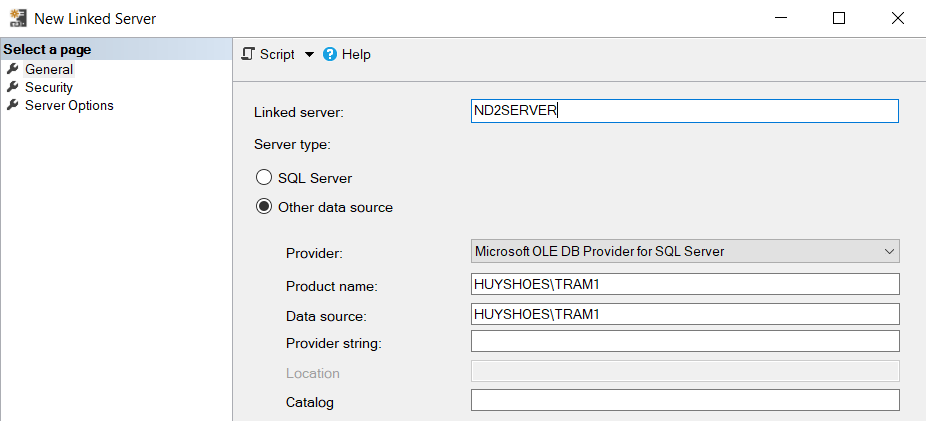


## **Connect**

Server Objects 🡪 Linked Server 🡪 New Linked Server...



Configure as follow:



Linked server: <Name trạm need to connect>

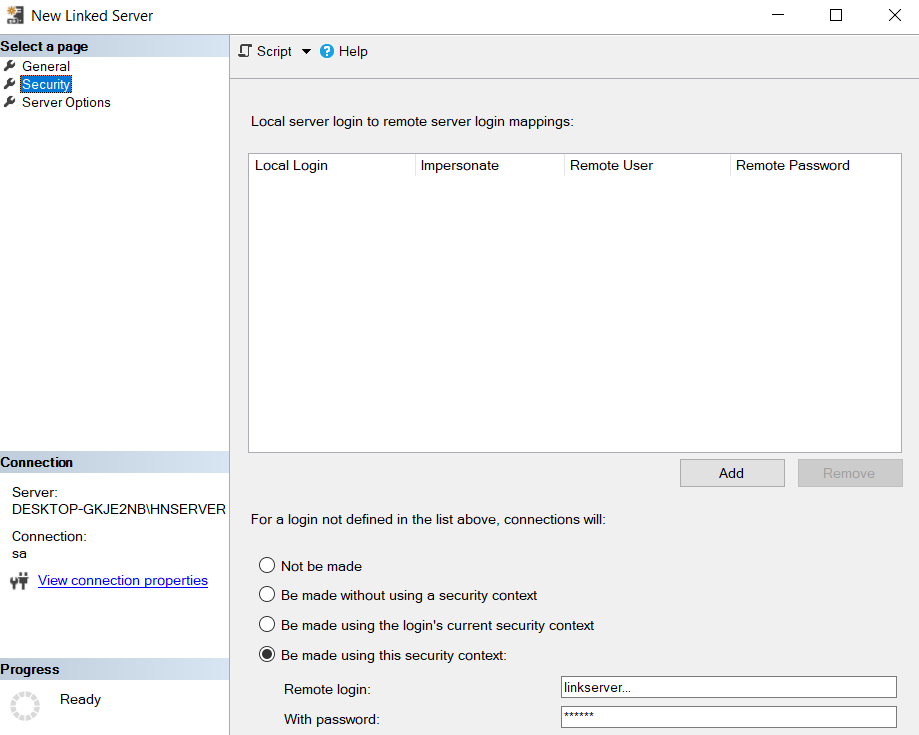
To ensure transparency when using Query, we set the Linked server as follow on all stations when connectiong

**Thai Binh:** TB1SERVER

**Nam Dinh:** ND1SERVER

**Ha Noi 2:** HN2SERVER

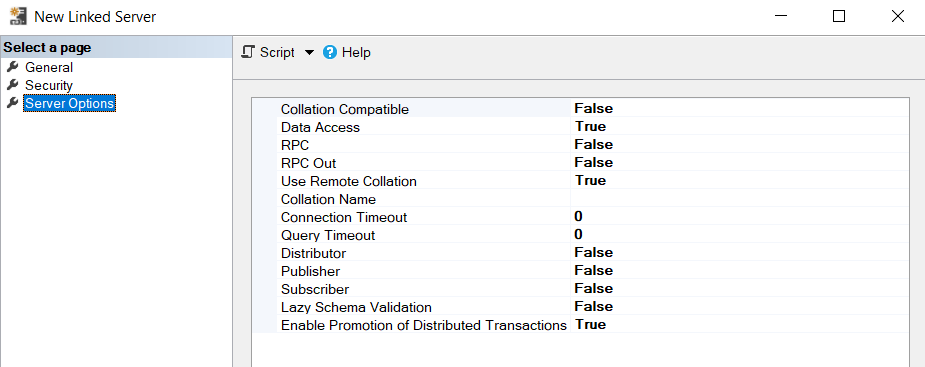
**Ha Noi:** HNSERVER



The server connects to the workstations using the sa. Account

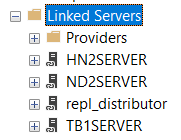
The workstation connects to other workstations using a read-only account (linkserver…)

Convert RPC and RPC out to true

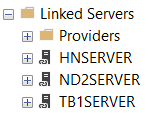


Click OK to connect

At server:



At client in Ha Noi 2:

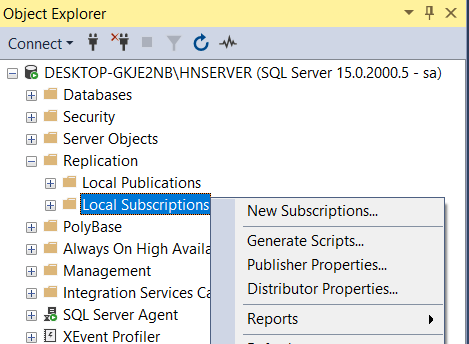


Do the same with other client

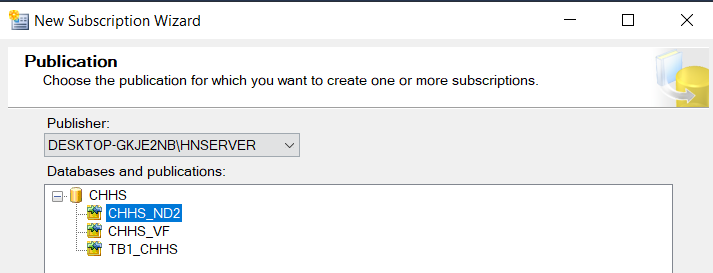
# **Distributed CSDL**

## **Create Subscriptions**

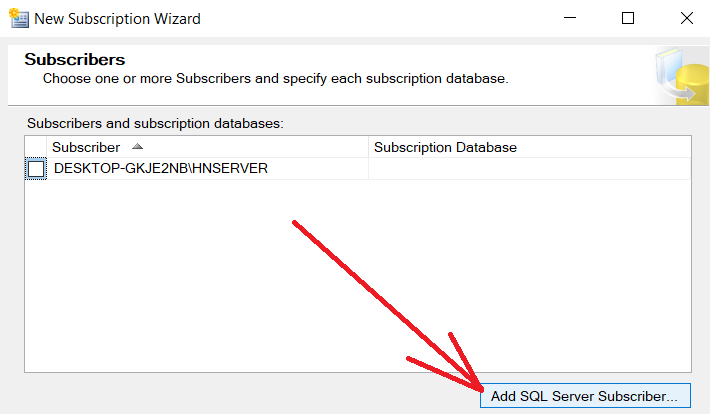
* Search for Replication, right click to Local Subcriptions, click new Subcriptions



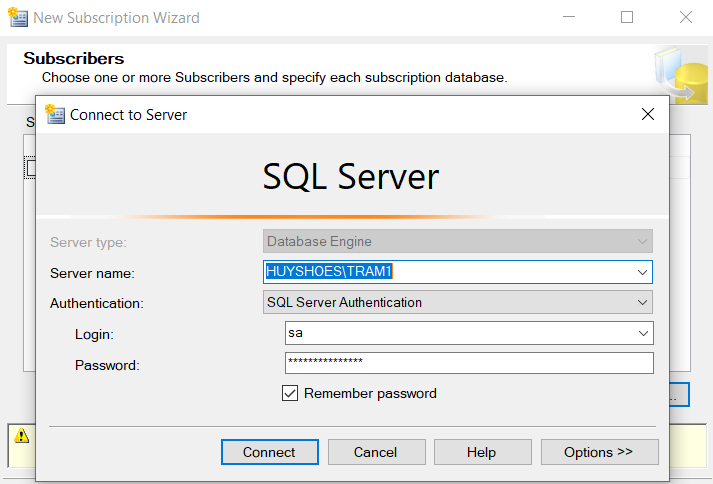
* Click next, chose Publication



Click “Add SQL ...”:

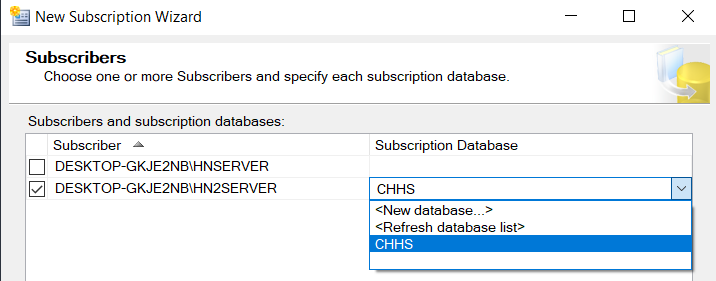


Login to Client need to connect:

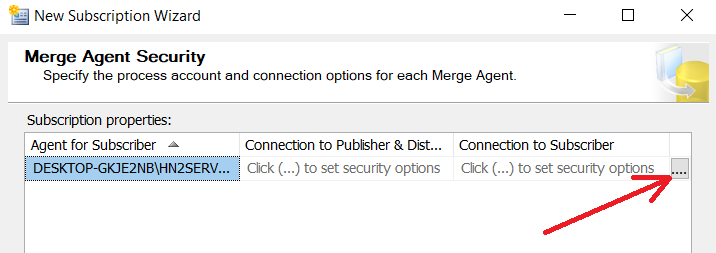


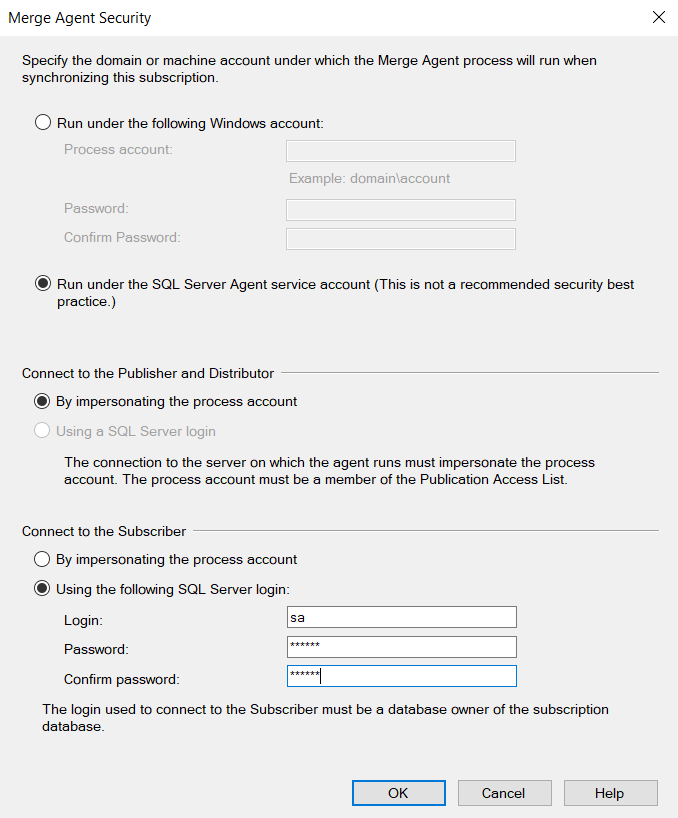
- At this step, you may encounter an error that cannot connect if you have not set up IP and port on the machines in SQL Server Configuration Manager. Connection errors can also occur if the machines cannot connect to each other via VPN. Or 2 servers on the same machine that are installed on the same port may also have an error.

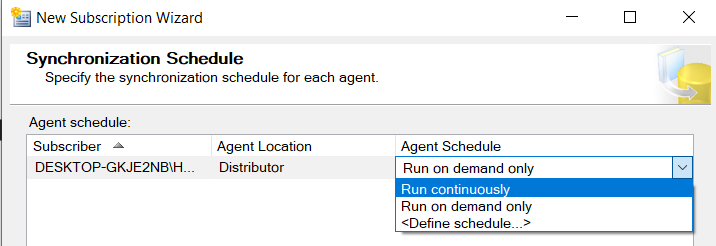
- Successful connection, select new database, name the newly created Database with the name of the Database you want to connect on the server:

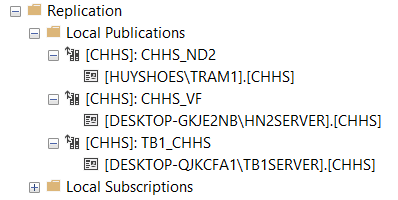


Choose as picture:





* Enter the server’s sa account, then click Next until the following table appears. Select Run continuouslt
* Click next until add Subscription complete

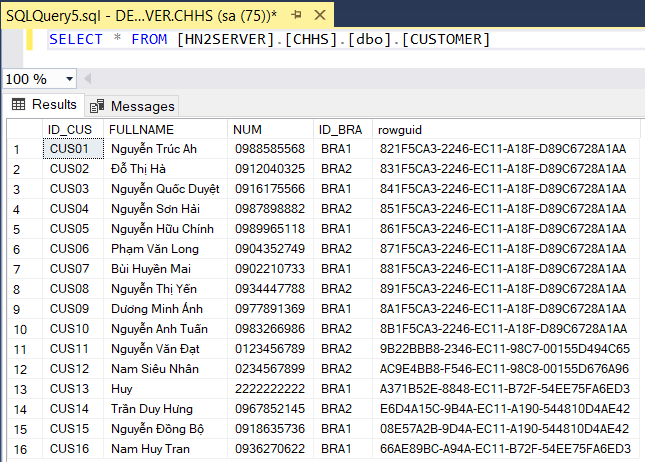


\* Note: After dispersal is complete. On stations, it is possible to reassign permissions to new databases in a read-only account, linkserver...

# **Query**

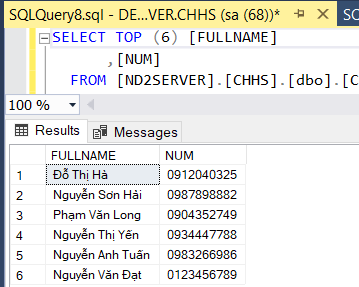
## **Show all Customer from Ha Noi 2 (HN2SERVER) from another branch**

|  |
| --- |
| SELECT \*  FROM [HN2SERVER].[CHHS].[dbo].[CUSTOMER] |



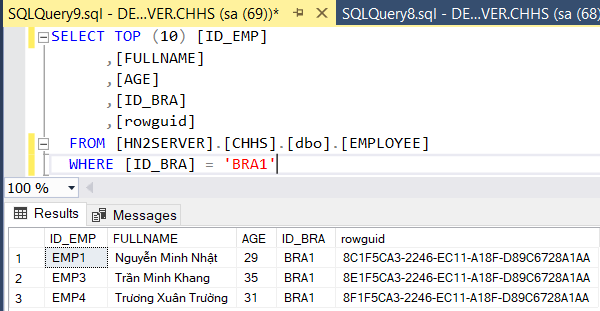
## **Show Name and Numer of top 10 Customer from Nam Dinh (ND2SERVER)**

|  |
| --- |
| SELECT TOP (6) [FULLNAME] ,[NUM]  FROM [ND2SERVER].[CHHS].[dbo].[CUSTOMER] |



## **Show top 10 Employee at Thai Binh from HN2SERVER**

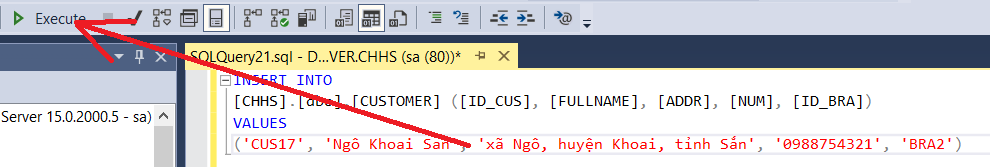
|  |
| --- |
| SELECT TOP (10) [ID\_EMP]  ,[FULLNAME]  ,[AGE]  ,[ID\_BRA]  ,[rowguid]  FROM [HN2SERVER].[CHHS].[dbo].[EMPLOYEE]  WHERE [ID\_BRA] = 'BRA1' |

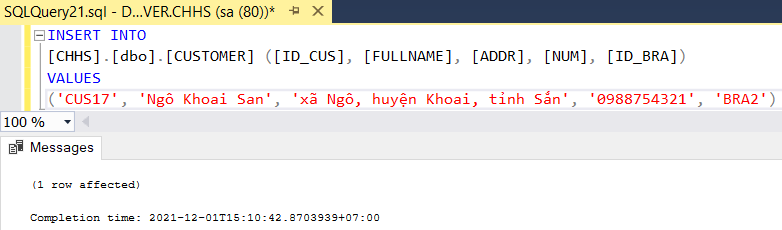


## **Insert data from Ha Noi (HNSERVER) and test the synchronization in Ha Noi 2 (HN2SERVER)**

|  |
| --- |
| INSERT INTO  [CHHS].[dbo].[CUSTOMER] ([ID\_CUS], [FULLNAME], [ADDR], [NUM], [ID\_BRA])  VALUES  ('CUS17', 'Ngô Khoai San', 'xã Ngô, huyện Khoai, tỉnh Sắn', '0988754321', 'BRA2') |

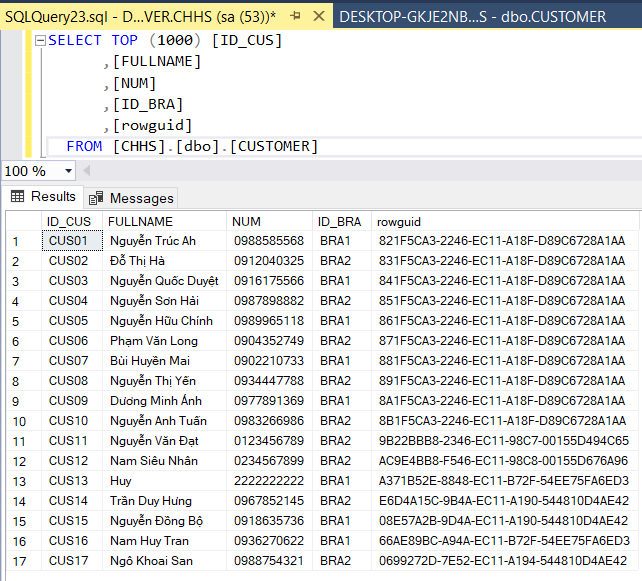
Click Execute:





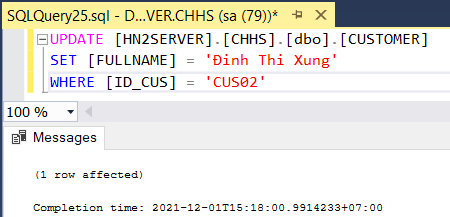
Just a moment... And check Ha Noi 2:

|  |
| --- |
| SELECT TOP (1000) [ID\_CUS]  ,[FULLNAME]  ,[NUM]  ,[ID\_BRA]  ,[rowguid]  FROM [CHHS].[dbo].[CUSTOMER] |



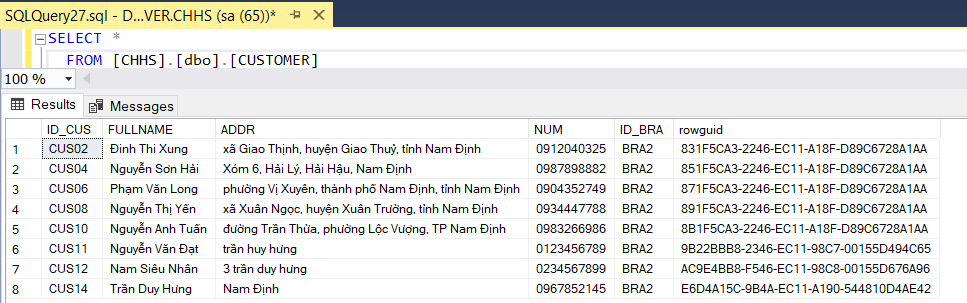
## **Update Customer from Ha Noi (HNSERVER) and test the synchronization**

|  |
| --- |
| UPDATE [HN2SERVER].[CHHS].[dbo].[CUSTOMER]  SET [FULLNAME] = 'Đinh Thi Xung'  WHERE [ID\_CUS] = 'CUS02' |

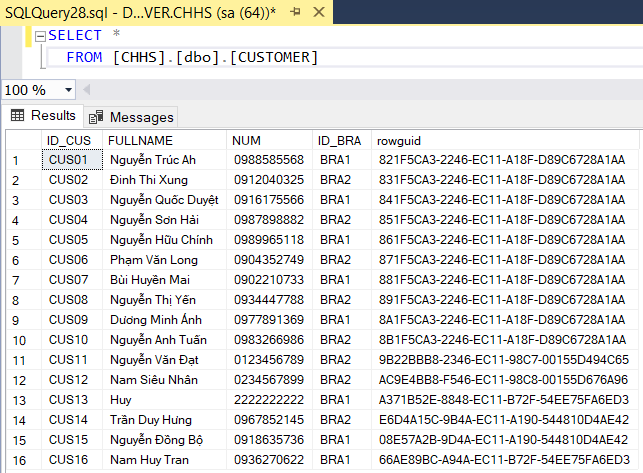


Check Customer from Nam Dinh (ND2SERVER):

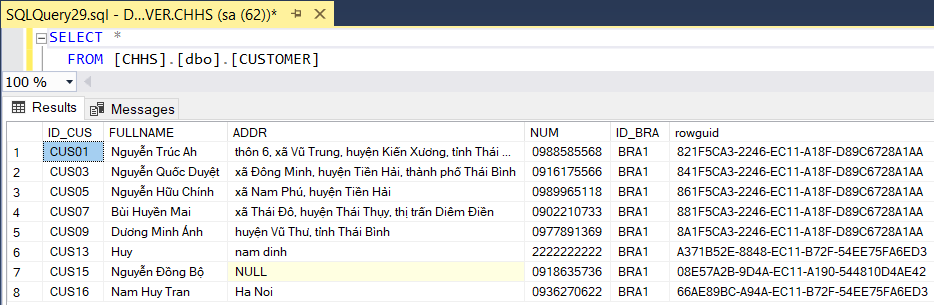
|  |
| --- |
| SELECT \*  FROM [CHHS].[dbo].[CUSTOMER] |



Check Customer from Ha Noi 2 (HN2SERVER):



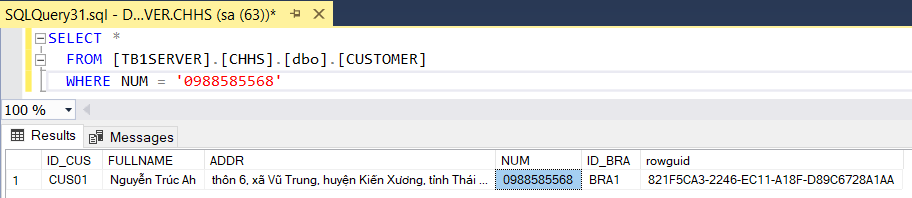
Check Customer from Thai Binh (TB1SERVER):



## **Find out a customer whose number phone is 0988585568 to test the Connection**

From Nam Dinh (ND2SERVER):

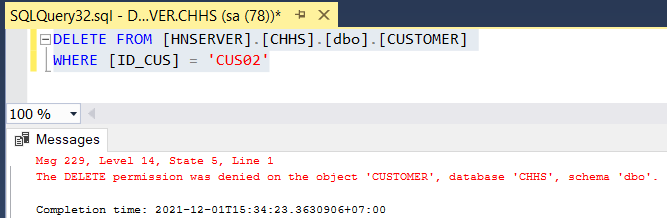
|  |
| --- |
| SELECT \*  FROM [TB1SERVER].[CHHS].[dbo].[CUSTOMER]  WHERE NUM = '0988585568' |



## **Test permission (Only Read)**

From Thai Binh (TB2SERVER):

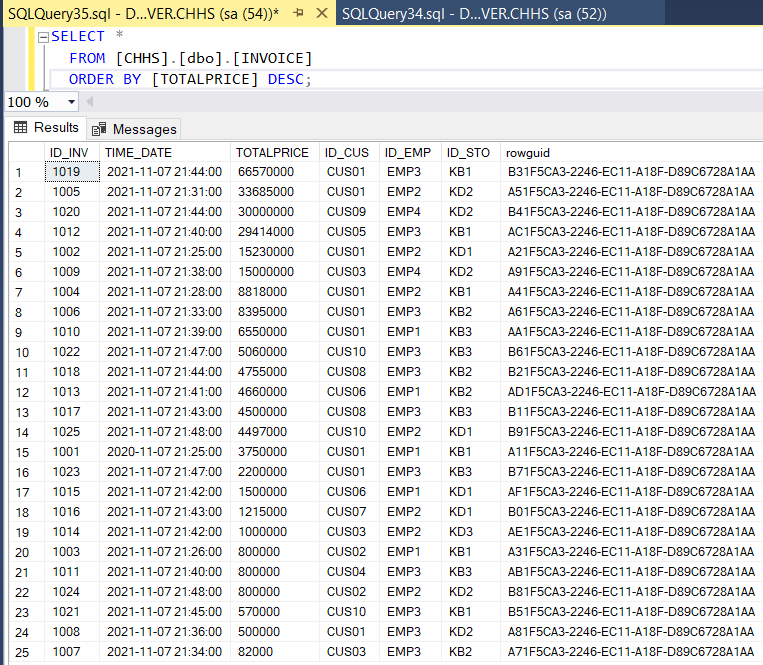
|  |
| --- |
| DELETE FROM [HNSERVER].[CHHS].[dbo].[CUSTOMER]  WHERE [ID\_CUS] = 'CUS02' |



* OK!!

## **Show all Invoice and Sort by Total Price**

|  |
| --- |
| SELECT \*  FROM [CHHS].[dbo].[INVOICE]  ORDER BY [TOTALPRICE] DESC; |



# **Concurrency & Commit Protocols**

## **Transaction**

The standard definition of a transaction states that “every query that runs is in a transaction,” that means any query we run is considered as being in a transaction. It could either be a simple SELECT query or any UPDATE or ALTER query.

* If we run a query without mentioning the BEGIN TRANSACTION keyword then it would be considered an implicit transition.
* If you run a query that starts with BEGIN TRANSACTION and ends with COMMIT or ROLLBACK, then it would be considered an explicit transaction.

A simple example:

|  |
| --- |
| -- 1. Start a Transaction  BEGIN TRANSACTION  -- 2. Change Customer Fullname  UPDATE [CHHS].[dbo].[CUSTOMER]  SET [FULLNAME] = 'Đinh Thị Cứng'  WHERE [ID\_CUS] = 'CUS02'  -- 3. Commit the DTC transaction  COMMIT TRANSACTION |

## **Commit Protocols**

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

The different distributed commit protocols are:

* One-phase commit
* Two-phase commit
* Three-phase commit

### ***One-phase Commit***

Distributed one-phase commit is the simplest commit protocol. The steps in distributed commit:

* After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site.
* The slaves wait for “Commit” or “Abort” message from the controlling site. This waiting time is called **window of vulnerability**.
* When the controlling site receives “DONE” message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
* On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

### ***Two-Phase Commit***

Two-phase commit (2PC) is a host server-installed protocol that ensures that updates to multiple instances of a database on a network either succeed or fail in their entirety.

Host Integration Server supports 2PC works using two components: the Microsoft Distributed Transaction Coordinator (DTC), and the transaction log. The DTC governs the normal DTC transaction flow: enlist, prepare, commit, and abort. Also, DTC handles transaction recovery in case of any failure or disconnection, while the transaction log maintains a log of information that is needed in case of recovery.

The steps performed in the ***Two-Phase***:

**Phase 1: Prepare Phase**

* After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
* The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
* A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

**Phase 2: Commit/Abort Phase**

* After the controlling site has received “Ready” message from all the slaves
  + The controlling site sends a “Global Commit” message to the slaves.
  + The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
  + When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
* After the controlling site has received the first “Not Ready” message from any slave
  + The controlling site sends a “Global Abort” message to the slaves.
  + The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  + When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

## **Concurrency control**

Concurrency is a situation that arises in a database due to the transaction process. Concurrency occurs when two or more than two users are trying to access the same data or information. DBMS concurrency is considered a problem because accessing data simultaneously by two different users can lead to inconsistent results or invalid behaviour

### ***Concurrency Problem Types***

* **Dirty Reads:** When another process reads the changed, but uncommitted data. This leads to the inconsistent state for the reader.
* **Lost Updates:** When two processes try to manipulate the same data simultaneously. This problem can lead to data loss, or the second process might overwrite the first process change.
* **Non-repeatable Reads:** when one process is reading the data, and another process is writing the data
* **Phantom Reads:** If two same queries executed by two users show different output. E.g. If A select a query to read some data, at the same time the B insert some new data but the A only get able to read the old data, when A re-query the same statement then he/she gets a different set of data

### ***Transaction isolation level***

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation level** | **Dirty reads** | **Nonrepeatable reads** | **Phantoms** |
| Read uncommitted | Y | Y | Y |
| Read committed | X | Y | Y |
| Repeatable read | X | X | Y |
| Serializable | X | X | X |

Y – May occur

X – Don’t occur

The following table describes simple ways that a DBMS might implement the transaction isolation levels.

### ***Two Phase Locking – 2PL***

A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases:

* **Growing Phase:** New locks on data items may be acquired but none can be released.
* **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

LOCK POINT - The Point at which the growing phase ends, i.e., when a transaction takes the final lock it needs to carry on its work

|  |  |  |  |
| --- | --- | --- | --- |
|  | **T1** | **T2** |  |
| **Growing Phase** | lock-S(A) |  | **Growing Phase** |
|  | lock-S(A) |
| lock-X(B)  **Lock Point** |  |
|  | ……. | …… |
| **Shrinking Phase** | Unlock(A) |  |
|  | Lock-X(C) **Lock Point** |  |
| Unlock(B) |  |
|  |  | Unlock(A) | **Shrinking Phase** |
|  | Unlock(C) |
| ……. | …… |  |

### ***Strict strong Two Phase Locking (SS2PL)***

SQL Server uses strong strict two-phase locking (SS2PL). It requires that the locks are only released after the transaction is finished and has been committed or rolled back. SS2PL provides serializability – database transactions appear as if they are atomic and occurring in complete isolation from one another.

## **Practise**

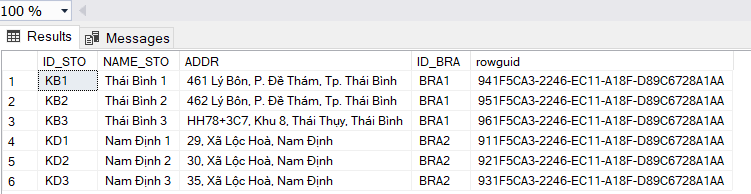
We use Serializable isolation level for all examples:

“SET TRANSACTION ISOLATION LEVEL SERIALIZABLE”

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

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

Use STORAGE table:



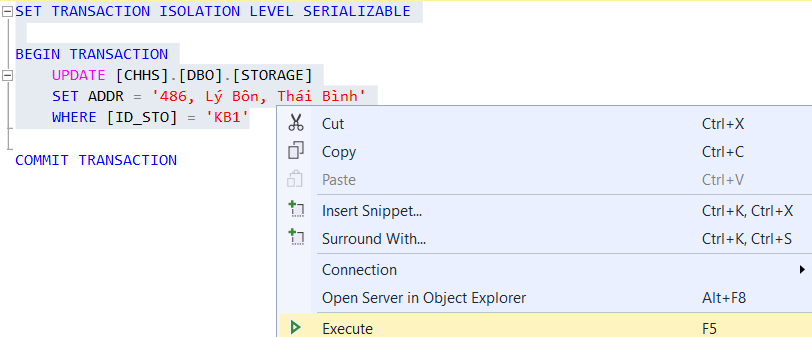
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

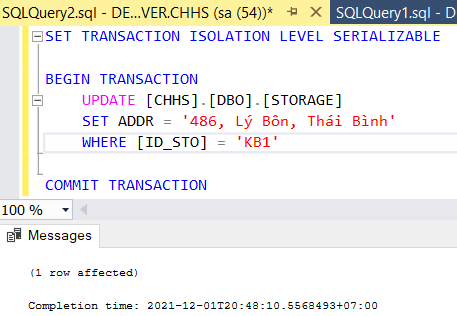
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '486, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

T2 – Show all STORAGE table

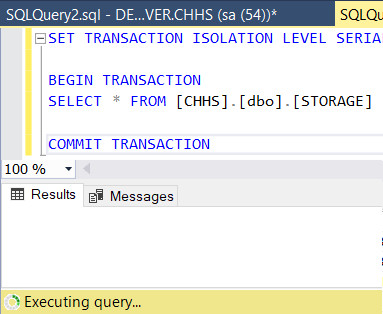
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  SELECT \* FROM [CHHS].[dbo].[STORAGE]  COMMIT TRANSACTION |

* Run T1 without committing it:



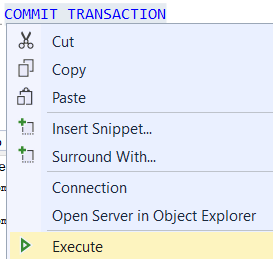


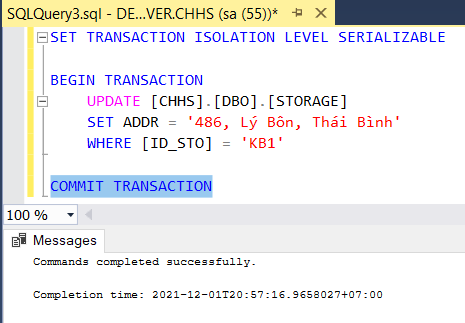
Run T2:



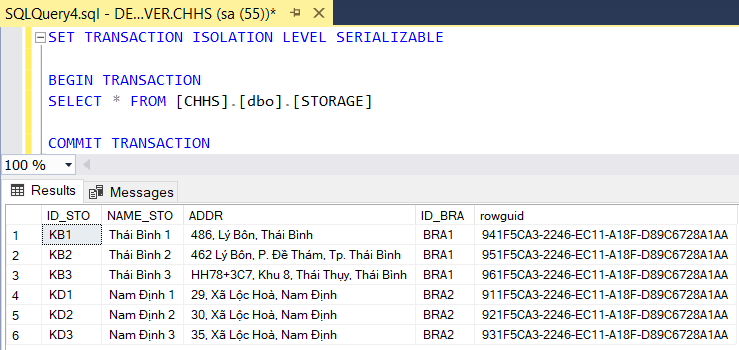
Nothing result (because of waiting for T1 to commit)

* Run COMMIT TRANSACTION:





Run T2 again:

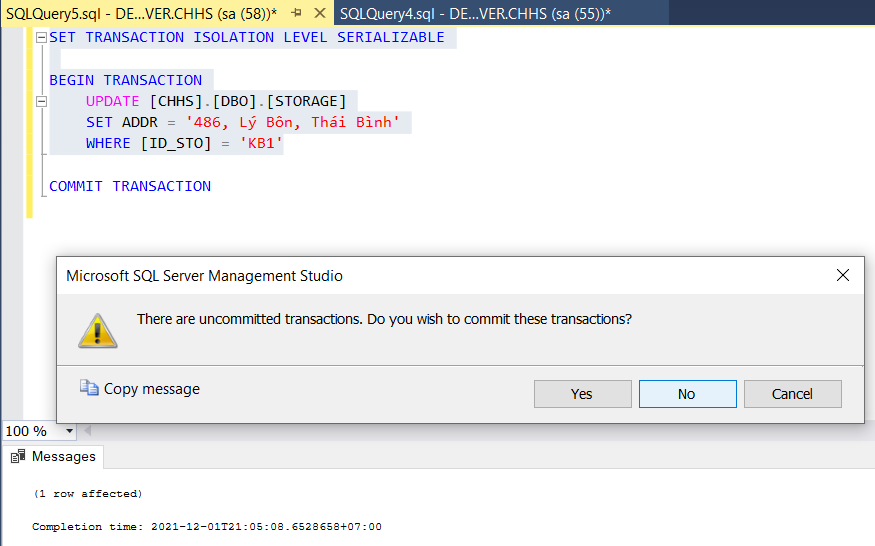


It runs immediately. Perfect! Because T1 has commited it

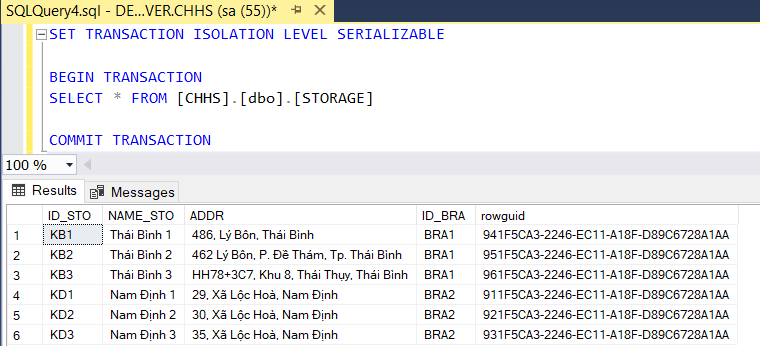
#### **+ CASE 2:** A aborted

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

Click No:



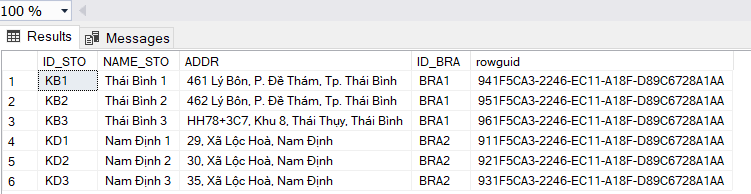
When we run T2, it’s completed immediately because the change from T1 doesn’t affect the current data ( it hasn’t yet been committed )



**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:



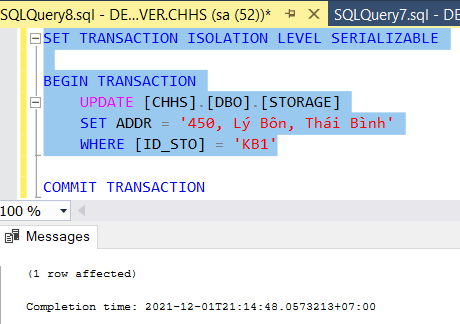
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

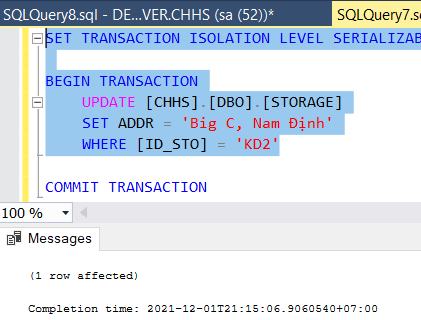
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '450, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

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

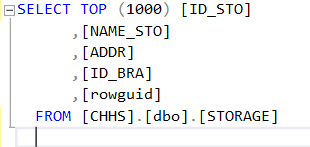
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = 'Big C, Nam Định'  WHERE [ID\_STO] = 'KD2'  COMMIT TRANSACTION |

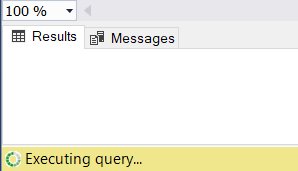
Run T1 & T2 without committing:



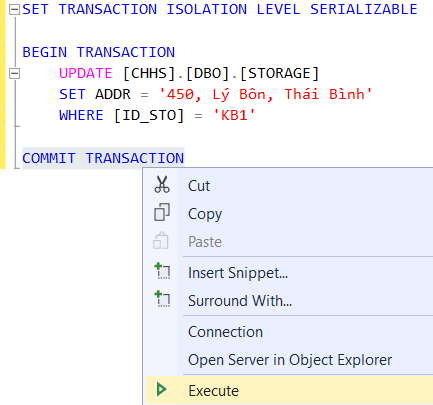


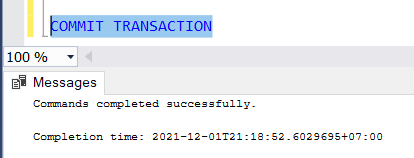
Select all STORAGE table:



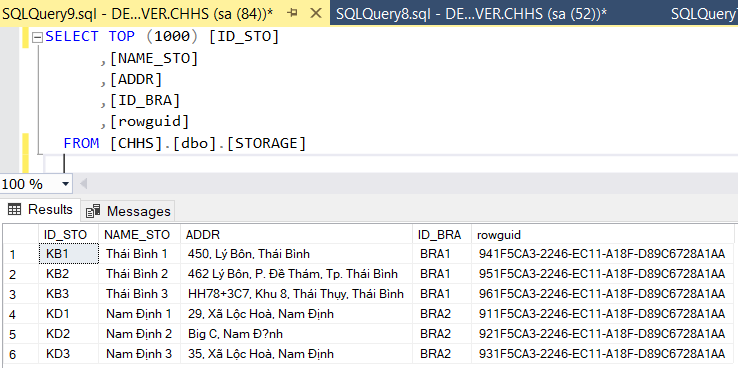


Now Commit T1 & T2:





Select all STORAGE table again:

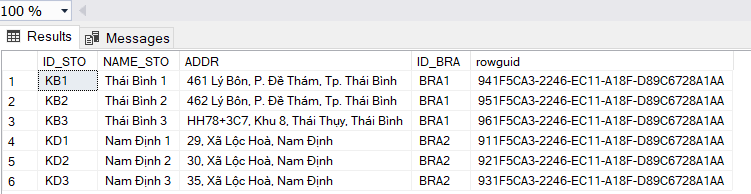


It’s completed immediately!

**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 |

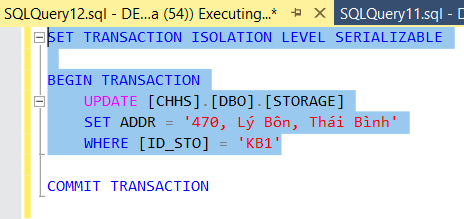
T2 – Write Address of Thái Bình 1 Storage (ID = KB1)

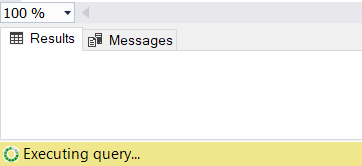
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '470, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

Run T1 without commiting it:

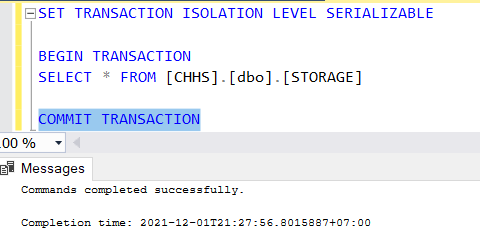


Run T2 without commiting it:





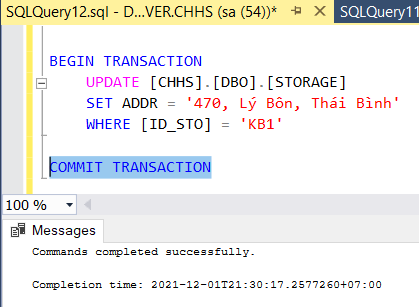
Now, Commit T1:



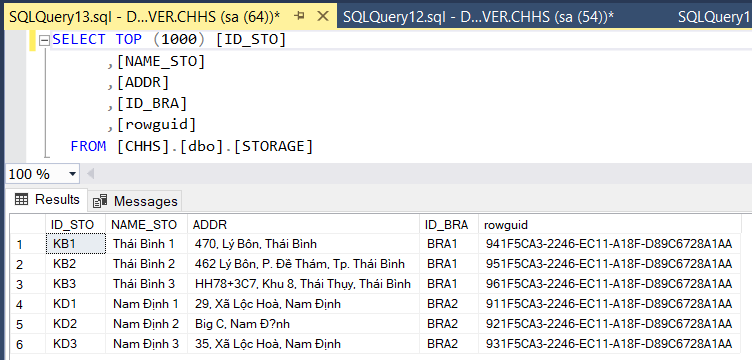
Then, move to T2 tab, we can see T2 has affected:



Now, Commit T2:



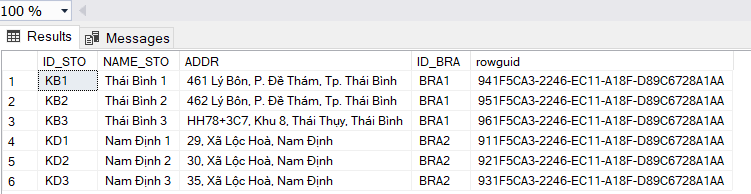
Finally, show STORAGE table:



It done!

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

We still use STORAGE table:



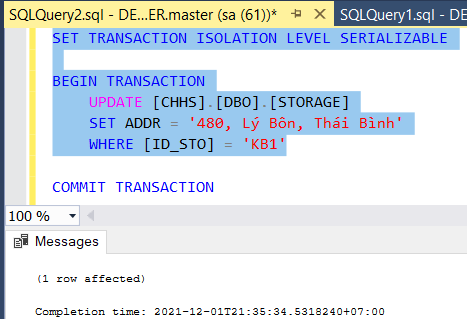
T1 – Write Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| 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)

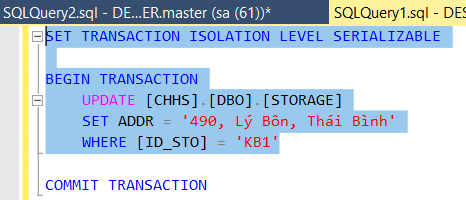
|  |
| --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  BEGIN TRANSACTION  UPDATE [CHHS].[DBO].[STORAGE]  SET ADDR = '490, Lý Bôn, Thái Bình'  WHERE [ID\_STO] = 'KB1'  COMMIT TRANSACTION |

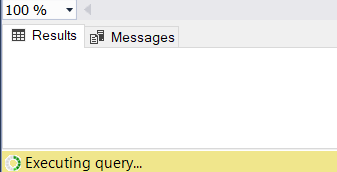
Run T1 without commiting it:



T1 is affected immediately!

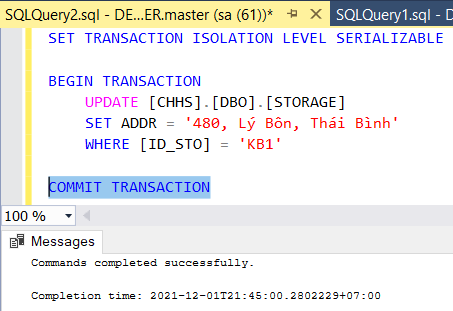
Run T2 without commiting it:



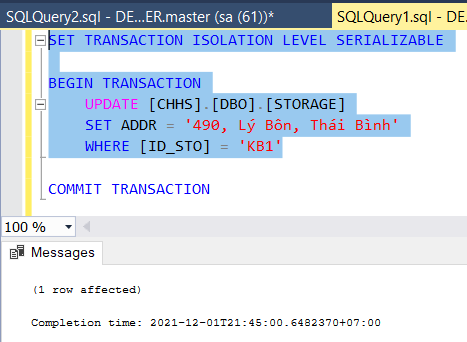


But T2 is not affected! (Because of waiting for T1)

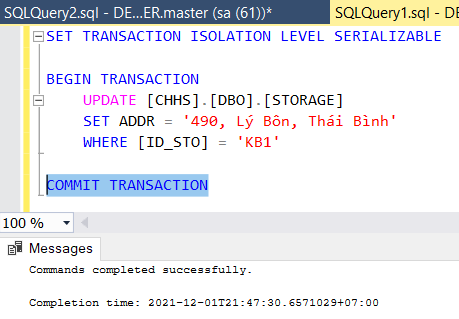
Now commiting T1:



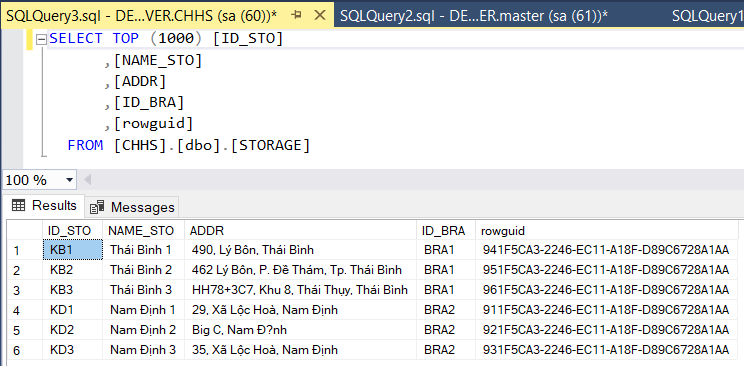
Then, move to T2 tab, we can see T2 has been affected:



Commiting T2:



Show STORAGE table:



## **Distributed Failure/Recovery**

### ***Types of failures:***

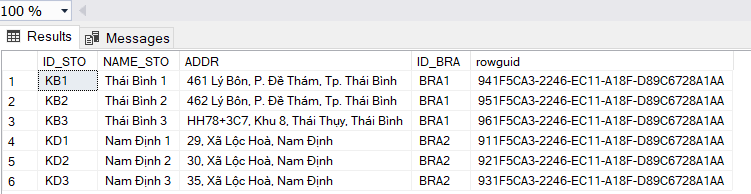
Different types of failures that may occur during the transaction:

1. **System crash:**  
   A hardware, software or network error comes under this category; these types of failures basically occur during the execution of the transaction. Hardware failures are basically considered as Hardware failures.
2. **System error:**  
   Some operation that is performed during the transaction is the reason for this type of error to occur, such as integer or divide by zero. This type of failure is also known as the transaction which may also occur because of erroneous parameter values or because of a logical programming error. In addition to this, the user may also interrupt the execution which may lead to failure in the transaction.
3. **Local error:**  
   This basically happens when we are doing the transaction but certain conditions may occur that may lead to cancellation of the transaction. This type of error is basically coming under Local error. The simple example of this is that data for the transaction may not be found. When we want to debit money from an insufficient balance account which leads to the cancellation of our request or transaction. And this exception should be programmed in the transaction itself so that it wouldn’t be considered as a failure.
4. **Concurrency control enforcement:**  
   The concurrency control method may decide to abort the transaction, to start again because it basically violates serializability or we can say that several processes are in a deadlock.
5. **Disk failure:**  
   This type of failure basically occurs when some disk loses their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read /write operation of the transaction.
6. **Catastrophe:**  
   These are also known as physical problems. It basically refers to the endless list of problems that include power failure or air-conditioning failure, fire, theft, sabotage, overwriting disk or tapes by mistake and mounting of the wrong tape by the operator.

The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques.

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

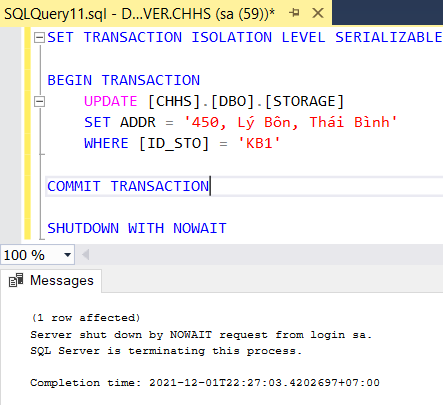
We still use STORAGE table:

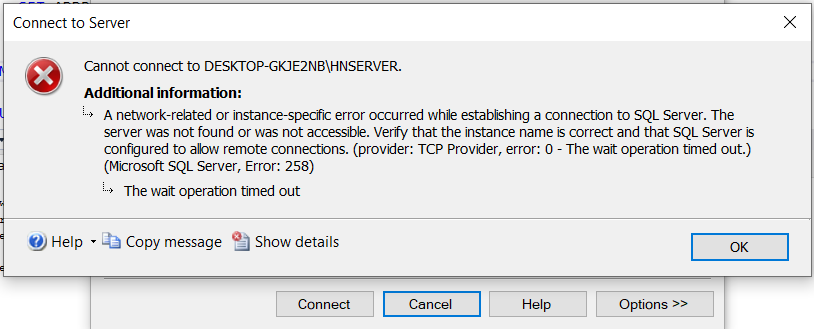


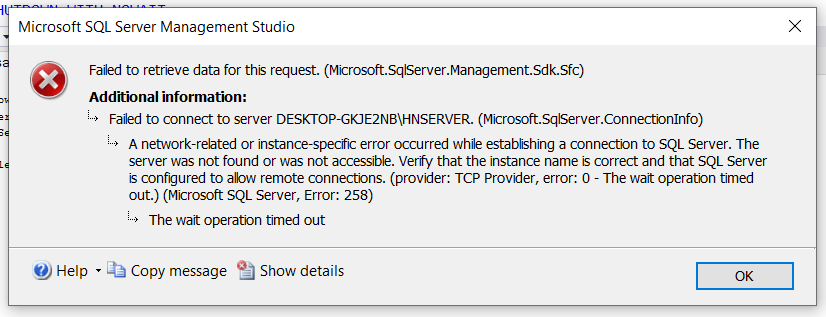
T1 – Update Address of Thái Bình 1 Storage (ID = KB1)

|  |
| --- |
| 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

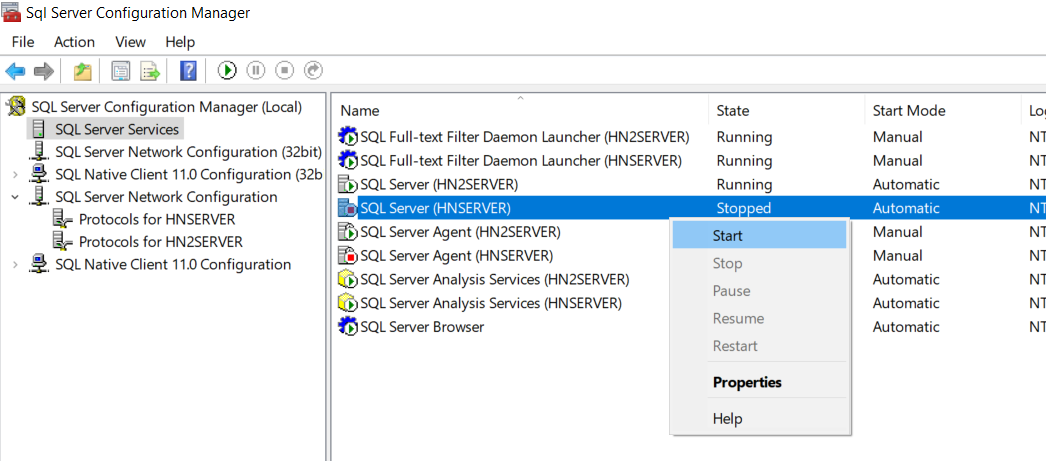


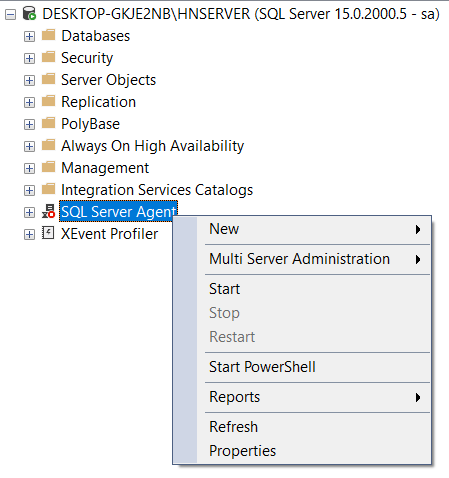




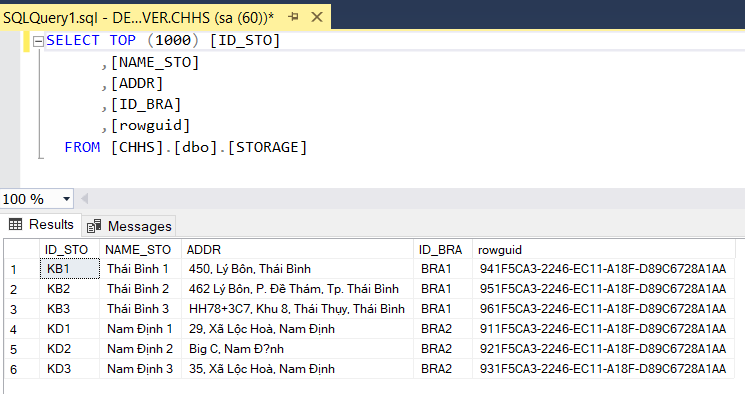
It’s stopped, we cannot refresh it, even!

We had to start it again:





Check Again:

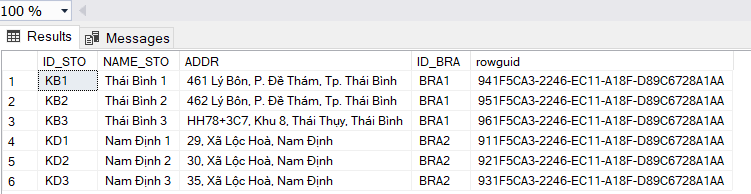


The change is committed successfully!

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

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

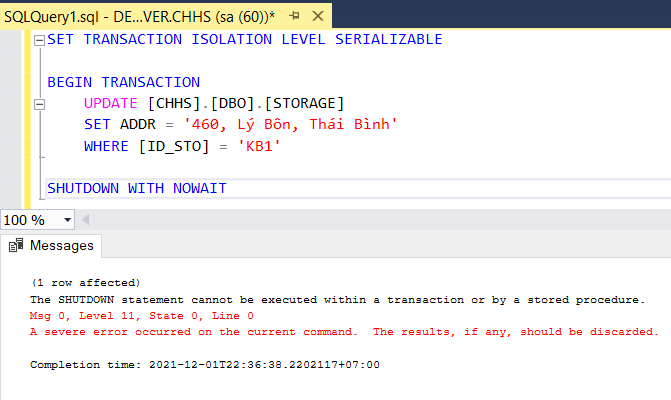
We still use STORAGE table:



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 |

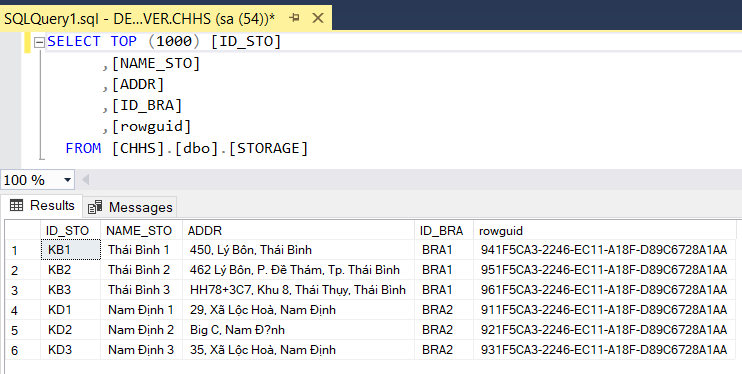
Run T1:



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:

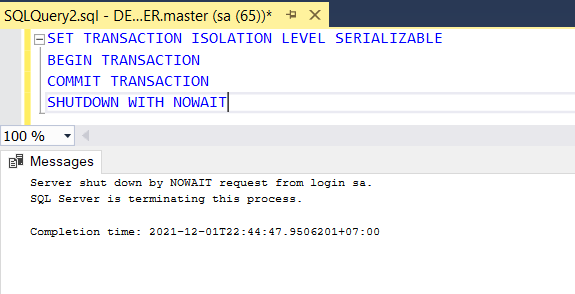


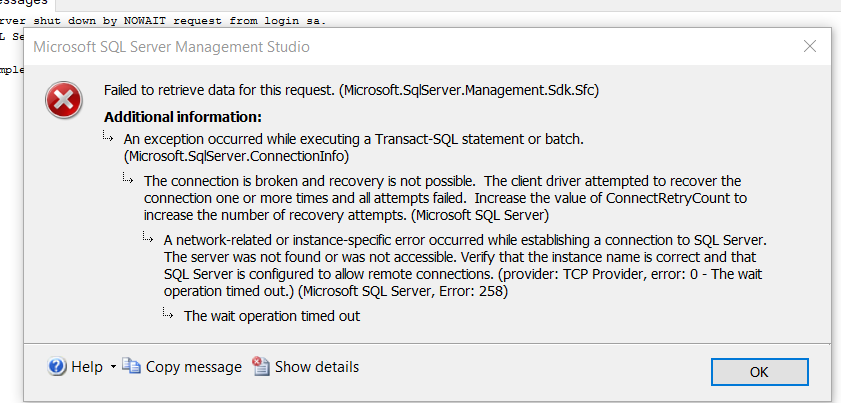
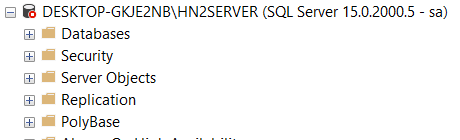
Nothing is changed!!

**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:



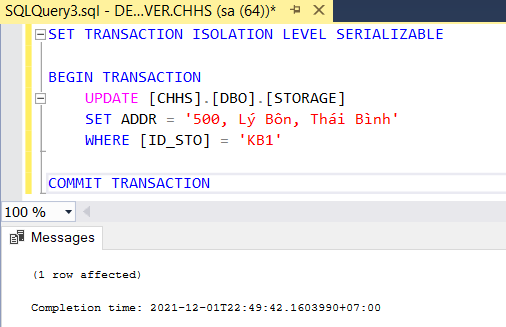
 

Now we will try to send transaction between other servers, suppose the master server update the STORAGE table:

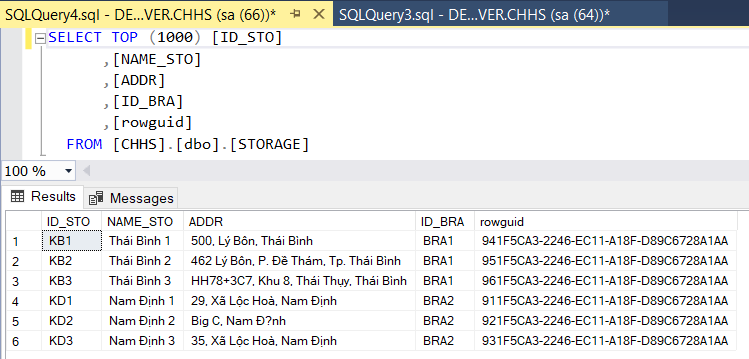
T2 – Update Address of Thái Bình 1 Storage (ID = KB1)

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

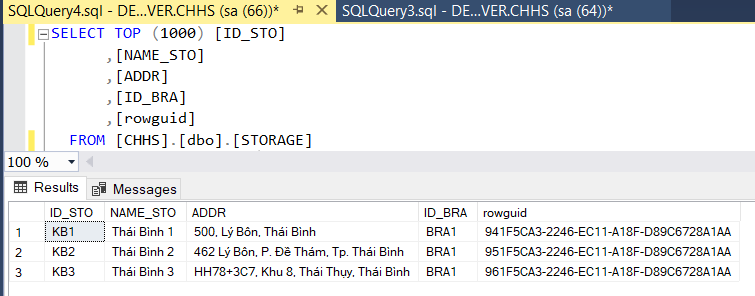
Run T2 on Ha Noi Server (master server):



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

# **References**

**Transaction:** <https://www.sqlshack.com/transactions-in-sql-server-for-beginners/>

**Two-phase:**

<https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_commit_protocols.htm>

<https://docs.microsoft.com/en-us/host-integration-server/core/how-to-perform-a-two-phase-commit-transaction-over-tcp-ip2>

**Currency:** <https://dotnettutorials.net/lesson/sql-server-concurrent-transactions/>

<https://www.geeksforgeeks.org/transaction-isolation-levels-dbms/?ref=lbphttps://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-ver15>

<https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-ver15>

**2PL:** <https://www.geeksforgeeks.org/two-phase-locking-protocol/>