Posts and Telecommunications Institute of Technology

Assignment

Distributed Database System

Subject: Distributed database system for motorbike manufacturing company management

Group: 1

Members

Nguyễn Minh Đức

Trương Xuân Dũng

Nguyễn Thanh Lâm

Content

Posts and Telecommunications Institute of Technology **1**
Group: 1 1
Members 1
Nguyễn Minh Đức 1
Trương Xuân Dũng 1
Nguyễn Thanh Lâm 1
Content **2**
Real-life scenario **8**
Scenario 8
Sites 8
Constraints 8
Entity Relationship Diagram **9**
Applications **9**
Horizontal Fragmentation **11**
Data fragmentation (app1-3) 11
Fragmenting table Staff 11
Predicate 11
Minterms 11
Check for correctness 11
Reconstruction 11
Disjointness 11
Completeness 12
Minimal 12
Conclusion 12
Derived fragmentation table Account 12
Semi-join 12
Completeness 12
Reconstruction 12
Disjointness 12
Conclusion 12
Derived fragmentation table Branch 13
Semi-join 13
Completeness 13
Reconstruction 13
Disjointness 13

Conclusion 13
Data fragmentation (app4-7) 13
Fragmenting table Branch 13
Predicate 13
Minterm 13
Check for correctness 14
Reconstruction 14
Disjointness 14
Completeness 14
Minimal 14
Conclusion 15
Derived fragmentation table Staff 15
Semi-join 15
Completeness 15
Reconstruction 15
Disjointness 15
Conclusion 15
Derived fragmentation table Account 15
Semi-join 15
Completeness 15
Reconstruction 16
Disjointness 16
Conclusion 16
Data fragmentation (app8-11) 16
Fragmenting table Branch 16
Derived fragmentation table Manager 16
Semi-join 16
Completeness 16
Reconstruction 16
Disjointness 16
Conclusion 17
Derived fragmentation table Account 17
Semi-join 17
Completeness 17
Reconstruction 17
Disjointness 17
Conclusion 17
Data Fragmentation (app 12-15) 17
Primary Fragmentation Branch 18

Derived Fragmentation table Stock 18
Semi-join 18
Completeness 18
Reconstruction 18
Disjointness 18
Conclusion 18
Data Fragmentation (app16-19) 18
Primary fragmentation Table Stock 18
Derived fragmentation Table StoreInStock 19
Semi-join 19
Completeness 19
Reconstruction 19
Disjointness 19
Conclusion 19
Derived fragmentation table Bike 19
Semi-join 19
Completeness 19
Reconstruction 19
Disjointness 20
Conclusion 20
Data Fragmentation (app20-23) 20
Primary fragmentation Table Stock 20
Derived fragmentation Table PartInStock 20
Semi-join 20
Completeness 20
Reconstruction 20
Disjointness 20
Conclusion 21
Derived fragmentation table Part 21
Semi-join 21
Completeness 21
Reconstruction 21
Disjointness 21
Conclusion 21
Data Fragmentation (app24-27) 21
Primary Fragmenting BikeOrder 21
Derived fragmentation Table BikeOrder 22
Semi-join 22
Completeness 22

Reconstruction 22
Disjointness 22
Conclusion 22
Data Fragmentation (app28-31) 22
Primary fragmentation Table BikeOrder 22
Derived fragmentation Table OrderBike 22
Semi-join 22
Completeness 23
Reconstruction 23
Disjointness 23
Conclusion 23
Derived fragmentation Table Bike 23
Semi-join 23
Completeness 23
Reconstruction 23
Disjointness 24
Conclusion 24
Data Fragmentation (app32-35) 24
Primary fragmentation Table Stock 24
Derived fragmentation Table PartOrder 24
Semi-join 24
Completeness 24
Reconstruction 24
Disjointness 24
Conclusion 24
Data Fragmentation (app36-39) 25
Primary fragmentation Table PartOrder 25
Derived fragmentation Table OrderPart 25
Semi-join 25
Completeness 25
Reconstruction 25
Disjointness 25
Conclusion 25
Derived fragmentation table Part 25
Semi-join 25
Completeness 26
Reconstruction 26
Disjointness 26
Conclusion 26

Vertical Fragmentation (Example) **26**
Applications 26
Data Fragmentation example(app1-3) 27
Cost matrix 27
Frequency matrix 27
Attribute relationship matrix (AA) 27
Clustered AA matrix (CA) 27
Place attribute: 27
Result 28
VF Algorithm 29
Conclusion 29

Implementation using SQLServer **29**
Notes 29
Setting up 29
Setup sa user 29
Get your ip info from “ip config” command 34
Port forwarding to open port on router 35
Find public ip using “what is my ip” to connect later 37
Open port “1433” via setting inbound, outbound rules 37
Enable TCP protocols using SQL Server Configuration Manager 43
Use query to create new tables, insert records to tables 44
Set up Virtual LAN network for our servers using VPN (tool: RadminVPN) 44
Add Publication 45
Add Subscription 59
Start Synchronization 78
Result 79

Practice with database **80**
Update data from Site4 and test the synchronization in Site1 (a subscription) 80
Query 81
Update data from Site1 and test the synchronization in Site4 82
Query 82
Insert data from Site4 and test the synchronization in Site1 (a subscription) 83
Query 84
Insert data from Site1 and test the synchronization in Site4 85
False adding procedure: 86
Correct adding procedure: 87
Delete data from Site4 and test the synchronization in Site1 (a subscription) 88

Query 89
Delete data from Site1 and test the synchronization in Site4 90
Query 91
Commit protocols/Concurrency control & Distributed Failure/Recovery 93
Commit protocols & Concurrency control 93
Commit protocols & Distributed 2-phase commit protocol 93
Commit protocols 93
Distributed Two-phase Commit 93
Concurrency control, 2PL & SS2PL 94
2-Phase Locking (2PL) 94
Strict Strong 2-Phase Locking (SS2PL) 95
Demo 95
Demonstrate that 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. 96
Demonstrate a delayed example: 96
Demonstrate an aborted example: 99
Demonstrate that two transactions operating on the same tables, but different rows, can execute concurrently. 100
Demonstrate that you handle read/write conflicts as well as write/write conflicts. 103
Read/write conflict 103
Write/write conflict 107
Prove that your scheme will not block indefinitely. I.e., if you have a deadlock, you must detect and correct for it. You may find that there are things that you don't handle well. As long as you demonstrate the above, it is okay if there are some things you can't handle - as long as you are aware of (and document) the conditions under which your concurrency control fails. 111
Distributed Failure/Recovery 115
Distributed Failure/Recovery 115
Demo 116
Correct recovery after a commit, to all sites having updates. 116
Correct recovery before a commit: transaction must be aborted at all sites. 119
At least some situations where failure is non-blocking (remaining sites can run transactions that do not involve the failed sites.) Your system does not need to be non-blocking in all cases, but you should document failures that can result in blocking or conditions that can result in an inconsistent database (hopefully you will have none of the above.) 122
User manual (how-to-use) 125

Data 125
Functions 126
Applications 126
How to use 127

References **127**

Real-life scenario

Scenario

The system of the motorbike manufacturing company can be used to manage essential informations of the company. Such as:

- Informations of bikes and parts of bike in stocks
- Orders for importing and exporting bikes and parts
- Informations of different branches, stocks in each branch.
- Informations of branch managers and staffs
- Accounts of system users, could be either managers or staffs

Sites

There are **4 sites** in total:

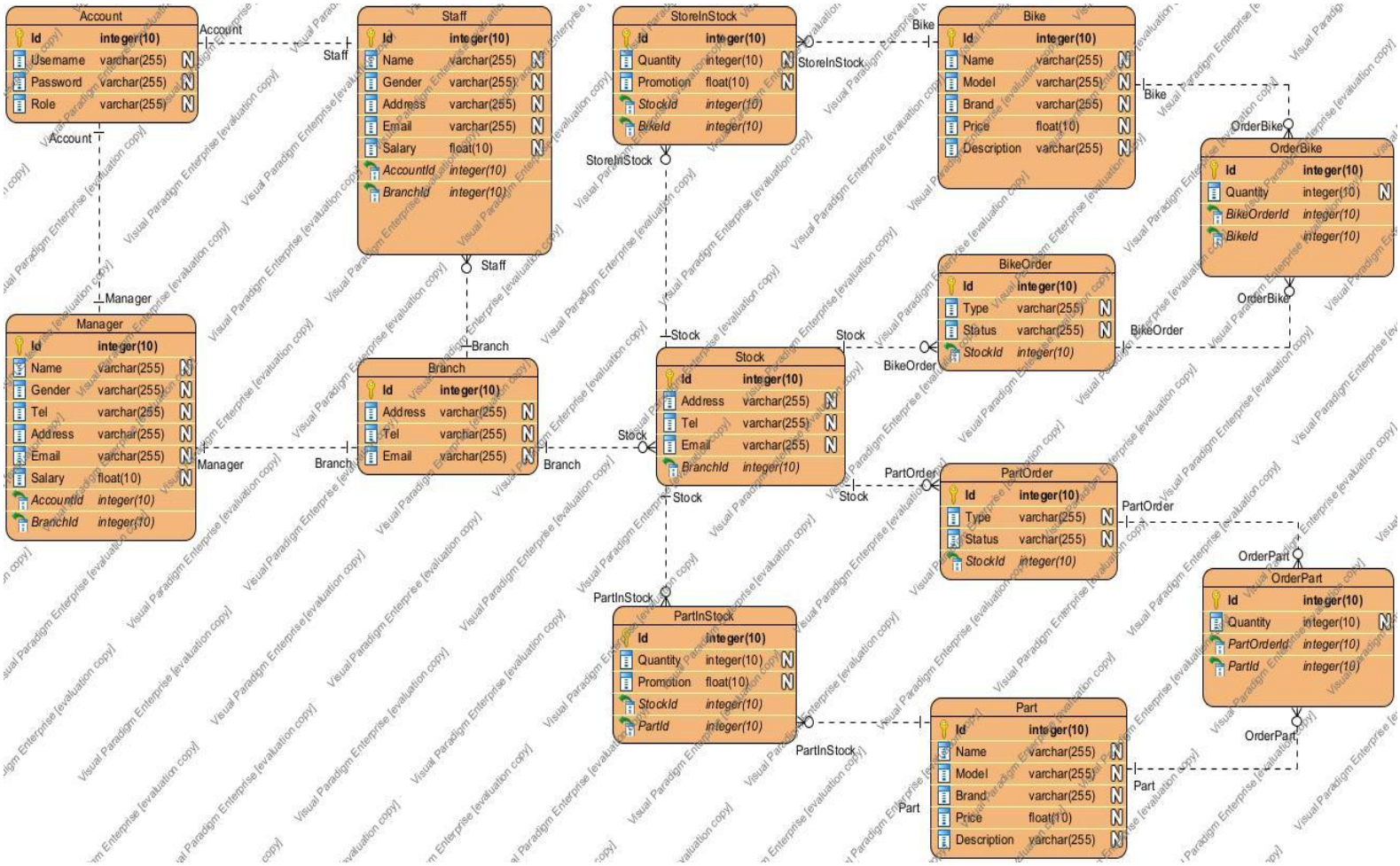
- 3 main sites based on locations: **Site1 - Hanoi**, **Site2 - Danang**, **Site 3 - Hochiminh**
- The only 1 site **(Site4)** which contains all information for all locations

Constraints

- There are 2 types of user: Branch Manager and Staff.
- Each Staff belongs to a Branch, a Branch has many Staffs.
- Each Manager manages only one Branch, every Branch is managed by only one Manager.
- Each user(Manager and Staff) has only one account to log in to the system, and each Account is owned by a user.
- Branches have one or many Stocks, Stocks belong to only 1 Branch
- Bikes can be stored in one or many Stocks, one Stock can store many Bikes
- Parts can be stored in one or many Stocks, one Stock can store many Parts
- Stocks can have 1 or many Order for Parts or Bikes, each Order belongs to 1 Stock

- Bikes or Parts can be in 1 or many BikeOrder/PartOrder, 1 Order can have many Bikes/Parts

Entity Relationship Diagram



Applications

1. List of staffs who have salary <500$
2. List of staffs who have salary >=500$ and <750$
3. List of staffs who have salary >= 750$
4. List of staffs in Hanoi's branch
5. List of staffs in Danang's branch
6. List of staffs in Hochiminh's branch

7. List of staffs that are not in Hanoi, Danang and Ho Chi Minh branch
8. List of managers in Hanoi's branch
9. List of managers in Danang's branch
10. List of managers in Hochiminh's branch
11. List of managers that are not in Hanoi, Danang and Ho Chi Minh branch
12. List of stocks in Hanoi's branch
13. List of stocks in Danang's branch
14. List of stocks in Hochiminh's branch
15. List of stocks that are not in Hanoi, Danang and Ho Chi Minh branch
16. List of bikes in Hanoi's stock
17. List of bikes in Danang's stock
18. List of bikes in Hochiminh's stock
19. List of bikes that are not in Hanoi, Danang and Ho Chi Minh branch
20. List of Parts in Hanoi branch stocks
21. List of Parts in Danang branch stocks
22. List of Parts in Ho Chi Minh branch stocks
23. List of Parts in not in Hanoi, Danang and Ho Chi Minh branch stocks
24. List of bike orders in stocks of Hanoi
25. List of bike orders in stocks of Danang
26. List of bike orders in stocks of Ho Chi Minh
27. List of bike orders that are not in Hanoi, Danang and Ho Chi Minh stocks
28. List of bikes in each Order in Hanoi branch stocks
29. List of bikes in each Order in Danang branch stocks
30. List of bikes in each Order in Hochiminh branch stocks
31. List of bikes in each Order that are not in Hanoi, Danang and Ho Chi Minh
32. List of part orders in Hanoi branch stocks
33. List of part orders in Danang branch stocks
34. List of part orders in Ho Chi Minh branch stocks
35. List of part orders that are not in Hanoi, Danang and Ho Chi Minh
36. List of Parts in each Order of Hanoi's Stocks
37. List of Parts in each Order of Danang's Stocks
38. List of Parts in each Order of Ho Chi Minh's Stocks
39. List of Parts in each Order that are not from Hanoi, Danang and Hochiminh stocks

Horizontal Fragmentation

1) Data fragmentation (app1-3)

Fragmenting table Staff

Predicate

1. p1 = salary <500$
2. p2 = salary >= 750$

Minterms

i	p(1)	p(2)	m(i)
1	1	1	∅
2	1	0	chọn
3	0	1	chọn
4	0	0	chọn

Acquired minterms are:

1. m1 = p1
2. m2 = p2
3. m3 = ¬p1 AND ¬p2

Check for correctness

<u>Reconstruction</u>

Fragment Si corresponds to mi(i <= 1 <= 4)

S1 U S2 U S3 = all records

<u>Disjointness</u>

Fragment Si corresponds to mi(i <= 1 <= 4)

S1 ∩ S2 ∩ S3 = ∅

Completeness

1. m1: app1, app2, app3 contain records with equal accessing probability
2. m2: app1, app2, app3 contain records with equal accessing probability
3. m3: app1, app2, app3 contain records with equal accessing probability

Minimal

1. m1 is used by at least app1
2. m2 is used by at least app3
3. m3 is used by at least app2

Conclusion

Table Staff can be divided into 3 fragments for application 1-3.

Derived fragmentation table Account

Semi-join

Account $\ltimes$ Si = 3 fragments of Staff

1. Account1 = Account $\ltimes$ S1
2. Account2 = Account $\ltimes$ S2
3. Account3 = Account $\ltimes$ S3

Completeness

Primary key Account.Id is not null, every Account.Id corresponds to Staff.AccountId

Reconstruction

Account1 U Account2 U Account3 = Account $\ltimes$ (S1 U S2 U S3) = Account $\ltimes$ Staff = Account (for Staff has satisfied completeness)

Disjointness

Account1 $\cap$ Account2 $\cap$ Account3 = Account $\ltimes$ (S1 $\cap$ S2 $\cap$ S3) = Account $\ltimes$ $\varnothing$ = $\varnothing$

Conclusion

Table Account can be divided into 3 corresponding fragments.

Derived fragmentation table Branch

Semi-join

Branch ⋉ Si = 3 fragments of Staff

1. Branch1 = Branch ⋉ S1
2. Branch2 = Branch ⋉ S2
3. Branch3 = Branch ⋉ S3

Completeness

Primary key Branch.Id is not null, every Branch.Id corresponds to Staff.BranchId

Reconstruction

Branch1 U Branch2 U Branch3 = Branch ⋉ (S1 U S2 U S3) = Branch ⋉ Staff = Branch (for Staff satisfied completeness)

Disjointness

Branch1 ∩ Branch2 ∩ Branch3 = Branch ⋉ (S1 ∩ S2 ∩ S3) = Branch ⋉ ∅ = ∅

Conclusion

Branch can be divided into 3 corresponding fragments.

2) Data fragmentation (app4-7)

Fragmenting table Branch

Predicate

1. p1: Branch.address = “Hanoi”
2. p2: Branch.address “Danang”
3. p3: Branch.address = “Hochiminh”

Minterm

i	p1	p2	p3	m(i)
1	1	1	1	∅

2	1	1	0	∅
3	1	0	1	∅
4	1	0	0	chọn
5	0	1	1	∅
6	0	1	0	chọn
7	0	0	1	chọn
8	0	0	0	chọn

The minterms:

1. m1 = p1 AND ¬p2 AND ¬p3
2. m2 = ¬p1 AND p2 AND ¬p3
3. m3 = ¬p1 AND ¬p2 AND p3
4. m4 = ¬p1 AND ¬p2 AND ¬p3

Check for correctness

Reconstruction

Si correspond to mi
S1 U S2 U S3 U S4 = all records of Branch

Disjointness

Si correspond to mi
S1 ∩ S2 ∩ S3 ∩ S4 = ∅

Completeness

1. m1: app4, app5, app6, app7 contain records with equal accessing probability
2. m2: app4, app5, app6, app7 contain records with equal accessing probability
3. m3: app4, app5, app6, app7 contain records with equal accessing probability
4. m4: app4, app5, app6, app7 contain records with equal accessing probability

Minimal

1. m1 is used by at least app4
2. m2 is used by at least app5
3. m3 is used by at least app6
4. m4 is used by at least app7

Conclusion

Branch can be divided into 4 fragments for application 4-7.

Derived fragmentation table Staff

Semi-join

Staff ⋉ Si = 4 fragments

1. Staff1 = Staff ⋉ S1
2. Staff2 = Staff ⋉ S2
3. Staff3 = Staff ⋉ S3
4. Staff4 = Staff ⋉ S4

Completeness

Primary key Staff.Id and foreign key Staff.BranchId is not null, every Staff.BranchId corresponds to Branch.Id.

Reconstruction

Staff1 U Staff2 U Staff3 U Staff4 = Staff ⋉ (S1 U S2 U S3 U S4) = Staff ⋉ Branch = Staff (for Branch satisfied completeness)

Disjointness

Staff1 ∩ Staff2 ∩ Staff3 ∩ Staff4 = Staff ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = Staff ⋉ ∅ = ∅

Conclusion

Staff can be divided into 4 corresponding fragments.

Derived fragmentation table Account

Semi-join

Account ⋉ Staff(i) = 4 fragments

1. Account1 = Account ⋉ Staff1
2. Account2 = Account ⋉ Staff2
3. Account3 = Account ⋉ Staff3
4. Account4 = Account ⋉ Staff4

Completeness

Primary key Account.Id is not null, every Account.Id corresponds to Staff.AccountId

Reconstruction

Account1 U Account2 U Account3 U Account4 = Account ⋉ (Staff1 U Staff2 U Staff3 U Staff4) = Account ⋉ Staff = Account (for Staff satisfied completeness)

Disjointness

Account1 ∩ Account2 ∩ Account3 ∩ Account4 = Branch ⋉ (Staff1 ∩ Staff2 ∩ Staff3 ∩ Staff4) = Account ⋉ ∅ = ∅

Conclusion

Table Branch can be divided into 4 corresponding fragments.

3) Data fragmentation (app8-11)

Fragmenting table Branch

Branch can be divided into 4 fragments (Finished in fragmentation from app4-7).

Derived fragmentation table Manager

Semi-join

Manager ⋉ Si = 4 fragments

5. Manager1 = Manager ⋉ S1
6. Manager2 = Manager ⋉ S2
7. Manager3 = Manager ⋉ S3
8. Manager4 = Manager ⋉ S4

Completeness

Primary key Manager.Id and foreign key Manager.BranchId is not null, every Manager.BranchId corresponds to Branch.Id.

Reconstruction

Manager1 U Manager2 U Manager3 U Manager4 = Manager ⋉ (S1 U S2 U S3 U S4) = Manager ⋉ Branch = Manager (for Branch satisfied completeness)

Disjointness

Manager1 ∩ Manager2 ∩ Manager3 ∩ Manager4 = Manager ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = Manager ⋉ ∅ = ∅

Conclusion

Manager can be divided into 4 corresponding fragments.

Derived fragmentation table Account

Semi-join

Account ⋉ Manager(i) = 4 fragments

5. Account1 = Account ⋉ Manager1
6. Account2 = Account ⋉ Manager2
7. Account3 = Account ⋉ Manager3
8. Account4 = Account ⋉ Manager4

Completeness

Primary key Account.Id is not null, every Account.Id corresponds to Manager.AccountId

Reconstruction

Account1 U Account2 U Account3 U Account4 = Account ⋉ (Manager1 U Manager2 U Manager3 U Manager4) = Account ⋉ Manager = Account (for Manager satisfied completeness)

Disjointness

Account1 ∩ Account2 ∩ Account3 ∩ Account4 = Branch ⋉ (Manager1 ∩ Manager2 ∩ Manager3 ∩ Manager4) = Account ⋉ ∅ = ∅

Conclusion

Table Branch can be divided into 4 corresponding fragments.

4) Data Fragmentation (app 12-15)

Primary Fragmentation Branch

Finished in fragmentation from app4-7

Derived Fragmentation table Stock

(each Si corresponds to Branch(i))

Semi-join

Branch ⋉ Si = 4 fragments

1. Stock1 = Stock ⋉ S1
2. Stock2 = Stock ⋉ S2
3. Stock3 = Stock ⋉ S3
4. Stock4 = Stock ⋉ S4

Completeness

Primary key Stock.Id is not null, every Branch.Id corresponds to Stock.BranchId

Reconstruction

Stock1 U Stock2 U Stock3 U Stock4 = Stock ⋉ (S1 U S2 U S3 U S4) = Stock ⋉ Branch = Stock (for Branch satisfied completeness)

Disjointness

Stock1 ∩ Stock2 ∩ Stock3 ∩ Stock4 = Stock ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = Stock ⋉ ∅ = ∅

Conclusion

Table Stock can be divided into 4 corresponding fragments.

4) Data Fragmentation (app16-19)

Primary fragmentation Table Stock

Finished in fragmentation from app8-11

Derived fragmentation Table StoreInStock

Semi-join

For Si from app8-11
StoreInStock ⋉ Si = 4 fragments

1. StoreInStock1 = StoreInStock ⋉ S1
2. StoreInStock2 = StoreInStock ⋉ S2
3. StoreInStock3 = StoreInStock ⋉ S3
4. StoreInStock4 = StoreInStock ⋉ S4

Completeness

Primary key StoreInStock.Id, forgein key BikeId, StockId is not null, every BikeId, StockId corresponds to Bike.Id, Stock.Id

Reconstruction

StoreInStock1 U StoreInStock2 U StoreInStock3 U StoreInStock4 = StoreInStock ⋉ (S1 U S2 U S3 U S4) = StoreInStock ⋉ Stock = StoreInStock (for Stock satisfied completeness)

Disjointness

StoreInStock1 ∩ StoreInStock2 ∩ StoreInStock3 ∩ StoreInStock4 = StoreInStock ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = StoreInStock ⋉ ∅ = ∅

Conclusion

Table StoreInStock can be divided into 4 corresponding fragments.

Derived fragmentation table Bike

<u>Semi-join</u>

Bike ⋉ StoreInStock(i) = 4 fragments

1) Bike1 = Bike ⋉ SIS1
2) Bike2 = Bike ⋉ SIS2
3) Bike3 = Bike ⋉ SIS3
4) Bike4 = Bike ⋉ SIS4

<u>Completeness</u>

Primary key Bike.Id is not null, every SIS.BikeId corresponds to Bike.Id

Reconstruction

Bike1 U Bike2 U Bike3 U Bike4 = Bike ⋉ (SIS1 U SIS2 U SIS3 U SIS4) = Bike ⋉ StoreInStock = Bike (for StoreInStock satisfied completeness)

Disjointness

Bike1 ∩ Bike2 ∩ Bike3 ∩ Bike4 = Bike ⋉ (SIS1 ∩ SIS2 ∩ SIS3 ∩ SIS4) = Bike ⋉ ∅ = ∅

Conclusion

Table Bike can be divided into 4 corresponding fragments.

5) Data Fragmentation (app20-23)

Primary fragmentation Table Stock

Finished in fragmentation from app8-11

Derived fragmentation Table PartInStock

Semi-join

With Si(1<=i<=4) from app8-11
PartInStock ⋉ Si = 4 fragments

1. PartInStock1 = PartInStock ⋉ S1
2. PartInStock2 = PartInStock ⋉ S2
3. PartInStock3 = PartInStock ⋉ S3
4. PartInStock4 = PartInStock ⋉ S4

Completeness

Primary key PartInStock.Id, forgein key PartId, StockId is not null, every PartId, StockId corresponds to Part.Id, Stock.Id

Reconstruction

PartInStock1 U PartInStock2 U PartInStock3 U PartInStock4 = PartInStock ⋉ (S1 U S2 U S3 U S4) = PartInStock ⋉ Stock = PartInStock (for Stock satisfied completeness)

Disjointness

PartInStock1 ∩ PartInStock2 ∩ PartInStock3 ∩ PartInStock4 = PartInStock ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = PartInStock ⋉ ∅ = ∅

Conclusion

Table PartInStock can be divided into 4 corresponding fragments.

Derived fragmentation table Part

Semi-join

Part ⋉ PartInStock(i) = 4 fragments

1. Part1 = Part ⋉ PIS1
2. Part2 = Part ⋉ PIS2
3. Part3 = Part ⋉ PIS3
4. Part4 = Part ⋉ PIS4

Completeness

Primary key Part.Id is not null, every PIS.PartId corresponds to Bike.Id

Reconstruction

Part1 U Part2 U Part3 U Part4 = Part ⋉ (S1 U S2 U S3 U S4) = Part ⋉ PartInStock = Part (for PartInStock satisfied completeness)

Disjointness

Part1 ∩ Part2 ∩ Part3 ∩ Part4 = Part ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = Part ⋉ ∅ = ∅

Conclusion

Table Part can be divided into 4 corresponding fragments.

6) Data Fragmentation (app24-27)

Primary Fragmenting BikeOrder

Finished in fragmentation from app 8-11

Derived fragmentation Table BikeOrder

Semi-join

With fragments Si from app8-11
BikeOrder ⋉ Si = 4 fragments

1. BikeOrder1 = BikeOrder ⋉ S1
2. BikeOrder2 = BikeOrder ⋉ S2
3. BikeOrder3 = BikeOrder ⋉ S3
4. BikeOrder4 = BikeOrder ⋉ S4

Completeness

Primary key: BikeOrder.Id, and Foreign key BikeOrder.StockId is not null, any BikeOrder.StockId equivalent to Stock.Id in Table Stock

Reconstruction

Si (i = 1,2,3,4) are fragments of previously fragmented Table Stock
BikeOrder1 U BikeOrder2 U BikeOrder3 U BikeOrder4 = BikeOrder ⋉ (S1 U S2 U S3 U S4) = BikeOrder ⋉ Stock = BikeOrder (for Stock satisfied completeness)

Disjointness

BikeOrder1 ∩ BikeOrder2 ∩ BikeOrder3 ∩ BikeOrder4 = BikeOrder ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = BikeOrder ⋉ ∅ = ∅

Conclusion

Table BikeOrder can be divided into 4 corresponding fragments.

7) Data Fragmentation (app28-31)

Primary fragmentation Table BikeOrder

Finished in fragmentation from app20-23

Derived fragmentation Table OrderBike

Semi-join

OrderBike ⋉ BikeOrder(i) = 4 fragments

1. OrderBike1 = OrderBike ⋉ BikeOrder1

2. OrderBike2 = OrderBike ⋉ BikeOrder2
3. OrderBike3 = OrderBike ⋉ BikeOrder3
4. OrderBike4 = OrderBike ⋉ BikeOrder4

Completeness

Primary key OrderBike.Id is not null, every OrderBike.BikeOrderId corresponds to BikeOrder.Id

Reconstruction

Si (i = 1,2,3,4) are fragments
OrderBike1 U OrderBike2 U OrderBike3 U OrderBike4 = OrderBike ⋉ (BikeOrder1 U BikeOrder2 U BikeOrder3 U BikeOrder4) = OrderBike ⋉ BikeOrder = OrderBike (for BikeOrder satisfied completeness)

Disjointness

OrderBike1 ∩ OrderBike2 ∩ OrderBike3 ∩ OrderBike4 = OrderBike ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = OrderBike ⋉ ∅ = ∅

Conclusion

Table OrderBike can be divided into 4 corresponding fragments.

Derived fragmentation Table Bike

Semi-join

Si (i = 1,2,3,4) are fragments of previously fragmented Table OrderBike
Bike ⋉ Si = 4 fragments
Bike1 = Bike ⋉ S1
Bike2 = Bike ⋉ S2
Bike3 = Bike ⋉ S3
Bike4 = Bike ⋉ S4

Completeness

Primary key Bike.Id and forgein key OrderBike.BikeId is not null, every Bike.Id corresponds to OrderBike.BikeId

Reconstruction

Si (i = 1,2,3,4) are fragments of previously fragmented Table OrderBike

Bike1 U Bike2 U BikeOrder3 U BikeOrder4 = Bike ⋉ (S1 U S2 U S3 U S4) = Bike ⋉ OrderBike = Bike (for OrderBike satisfied completeness)

Disjointness

Bike1 ∩ Bike2 ∩ Bike3 ∩ Bike4 = Bike ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = Bike ⋉ ∅ = ∅

Conclusion

Table Bike can be divided into 4 corresponding fragments.

8) Data Fragmentation (app32-35)

Primary fragmentation Table Stock

Finished in fragmentation from app8-11

Derived fragmentation Table PartOrder

Semi-join

With fragments Si from app8-11
PartOrder ⋉ Si = 4 fragments

1. PartOrder1 = PartOrder ⋉ S1
2. PartOrder2 = PartOrder ⋉ S2
3. PartOrder3 = PartOrder ⋉ S3
4. PartOrder4 = PartOrder ⋉ S4

Completeness

Primary key PartOrder.Id, and forgein key PartOrder.StockId is not null, every PartOrder.StockId corresponds to Stock.Id in Table Stock

Reconstruction

Si (i = 1,2,3,4) are fragments of previously fragmented Table Stock
PartOrder1 U PartOrder2 U PartOrder3 U PartOrder4 = PartOrder ⋉ (S1 U S2 U S3 U S4) = PartOrder ⋉ Stock = PartOrder (for Stock satisfied completeness)

Disjointness

PartOrder1 ∩ PartOrder2 ∩ PartOrder3 ∩ PartOrder4 = PartOrder ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = PartOrder ⋉ ∅ = ∅

Conclusion

Table PartOrder can be divided into 4 corresponding fragments.

9) Data Fragmentation (app36-39)

Primary fragmentation Table PartOrder

Finished in fragmentation from app28-31.

Derived fragmentation Table OrderPart

Semi-join

OrderPart ⋉ PartOrder(i) = 4 fragments

1. OrderPart1 = OrderPart ⋉ PartOrder1
2. OrderPart2 = OrderPart ⋉ PartOrder2
3. OrderPart3 = OrderPart ⋉ PartOrder3
4. OrderPart4 = OrderPart ⋉ PartOrder4

Completeness

Primary key OrderPart.Id is not null, every OrderPart.PartOrderId corresponds to PartOrder.Id

Reconstruction

Si (i = 1,2,3,4) are fragments
OrderPart1 U OrderPart2 U OrderPart3 U OrderPart4 = OrderPart ⋉ (PartOrder1 U PartOrder2 U PartOrder3 U PartOrder4) = OrderPart ⋉ PartOrder = OrderPart (for PartOrder satisfied completeness)

<u>Disjointness</u>

OrderPart1 ∩ OrderPart2 ∩ OrderPart3 ∩ OrderPart4 = OrderPart ⋉ (S1 ∩ S2 ∩ S3 ∩ S4) = OrderPart ⋉ ∅ = ∅

Conclusion

Table OrderPart can be divided into 4 corresponding fragments.

Derived fragmentation table Part

Semi-join

Si (i = 1,2,3,4)are fragments of previously fragmented Table OrderPart
Part $\ltimes$ Si = 4 fragments
Part1 = Part $\ltimes$ S1
Part2 = Part $\ltimes$ S2
Part3 = Part $\ltimes$ S3
Part4 = Part $\ltimes$ S4

Completeness

Primary key Part.Id and forgein key OrderPart.PartId is not null, every Part.Id corresponds to OrderPart.PartId

Reconstruction

Si (i = 1,2,3,4) are fragments of previously fragmented Table OrderPart
Part1 U Part2 U Part3 U Part4 = Part $\ltimes$ (S1 U S2 U S3 U S4) = Part $\ltimes$ OrderPart= Part (for Stock satisfied completeness)

Disjointness

Part1 $\cap$ Part2 $\cap$ Part3 $\cap$ Part4 = Part $\ltimes$ (S1 $\cap$ S2 $\cap$ S3 $\cap$ S4) = Part $\ltimes$ $\varnothing$ = $\varnothing$

Conclusion

Table Part can be divided into 4 corresponding fragments.

Vertical Fragmentation (Example)

1) Applications

Example for application1-3:
Our queries:

1. query1: Select name, email, salary
2. query2: Select name, gender, address, email, salary
3. query3: Select email, salary

2) Data Fragmentation example(app1-3)

Cost matrix

	A2	A3	A4	A5	A6
q1	1	0	0	1	1
q2	1	1	1	1	1
q3	0	0	0	1	1

Frequency matrix

	S1	S2	S3	S4	∑Si
q1	400	425	375	250	1450
q2	325	300	250	200	1075
q3	375	410	360	290	1435
					3960

Attribute relationship matrix (AA)

	A2	A3	A4	A5	A6
A2	2525	1075	1075	2525	2525
A3	1075	1075	1075	1075	1075
A4	1075	1075	1075	1075	1075
A5	2525	1075	1075	3960	3960
A6	2525	1075	1075	3960	3960

Clustered AA matrix (CA)

<u>Place attribute:</u>

A4

(Ai,Aj,Ak)	bond(Ai,Aj)	bond(Ai,Aj)	bond(Ai,Aj)	cont(Ai,Aj,Ak)
0,4,2	0	10454375	0	10454375
2,4,3	10454375	5778125	10454375	5778125
3,4,0	5778125	0	0	5778125

=> (A4,A2,A3)

A5

(Ai,Aj,Ak)	bond(Ai,Aj)	bond(Ai,Aj)	bond(Ai,Aj)	cont(Ai,Aj,Ak)
0,5,4	0	13539625	0	13539625
4,5,2	13539625	28684875	10454375	31770125
2,5,3	28684875	13539625	10454375	31770125
3,5,0	13539625	0	0	13539625

=> (A4,A5,A2,A3)

A6

(Ai,Aj,Ak)	bond(Ai,Aj)	bond(Ai,Aj)	bond(Ai,Aj)	cont(Ai,Aj,Ak)
0,6,4	0	13539625	0	13539625
4,6,5	13539625	40050075	13539625	40050075
5,6,2	40050075	28684875	28684875	40050075
2,6,3	28684875	13539625	10454375	31770125
3,6,0	13539625	0	0	13539625

=> (A4,A6,A5,A2,A3)

Result

	A4	A6	A5	A2	A3
A4	1075	1075	1075	1075	1075
A6	1075	3960	3960	2525	1075

A5	1075	3960	3960	2525	1075
A2	1075	2525	2525	2525	1075
A3	1075	1075	1075	1075	1075

VF Algorithm

	Case1	Case2	Case3	Case4
TA	A4	A4,A6	A4,A6,A5	A4,A6,A5,A2
BA	A6,A5,A2,A3	A5,A2,A3	A2,A3	A3
TQ	∅	∅	q3	q1,q3
BQ	q1,q3	∅	∅	∅
OQ	q2	q1,q2,q3	q1,q2	q2
CTQ	0	0	1435	2885
CBQ	2885	0	0	0
COQ	1075	3960	2525	1075
Z	-1155625	-15681600	-6375625	-1155625

Conclusion

The table of (staffs who have salary <500$) can be divided into 2 tables (**Id**,Address) and (**Id**,Name,Gender,Email,Salary)

Implementation using SQLServer

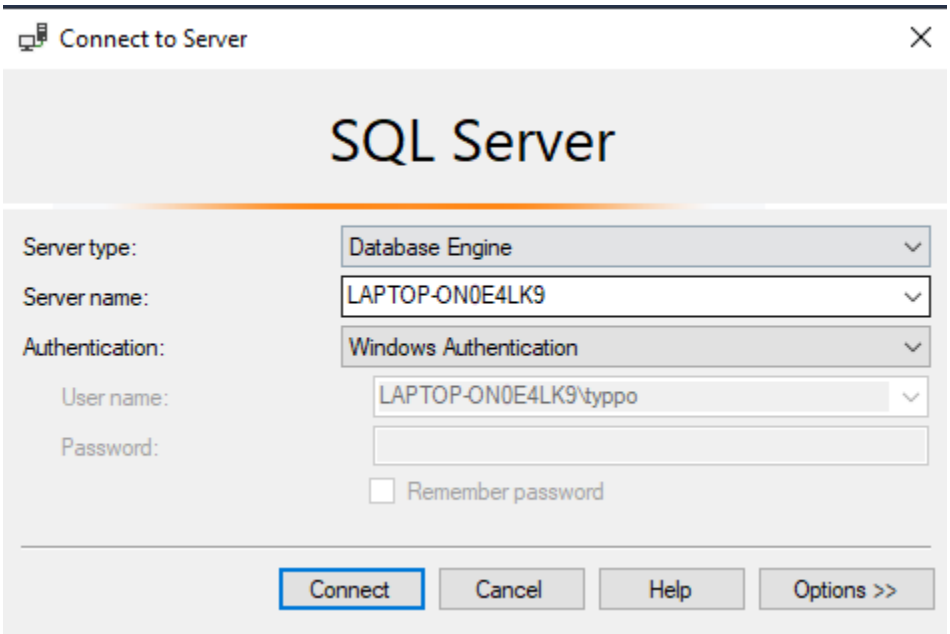
Notes

In this implementation we only use some of the previous horizontal fragmentations from application 4-6, 8-10, 12-14, 16-18, 20-22, 24-26, 28-30, 32-34, 36-38 (We dont fragment the Bike table and Part table for global uses from other branches)

Setting up

Setup sa user

Firstly, connect to SQL Server with Window Authentication



Security -> Logins -> sa -> properties, set the user like below

Login Properties - sa

Select a page
- General
- Server Roles
- User Mapping
- Status

Script Help

Login name: sa Search...

Windows authentication

SQL Server authentication

Password: ••••••••••••••••

Confirm password: ••••••••••••••••

Specify old password

Old password:

Enforce password policy

Enforce password expiration

User must change password at next login

Mapped to certificate

Mapped to asymmetric key

Map to Credential Add

Mapped Credentials

Credential	Provider

Remove

Default database: master

Default language: English - us_english

Connection

Server:
LAPTOP-ON0E4LK9

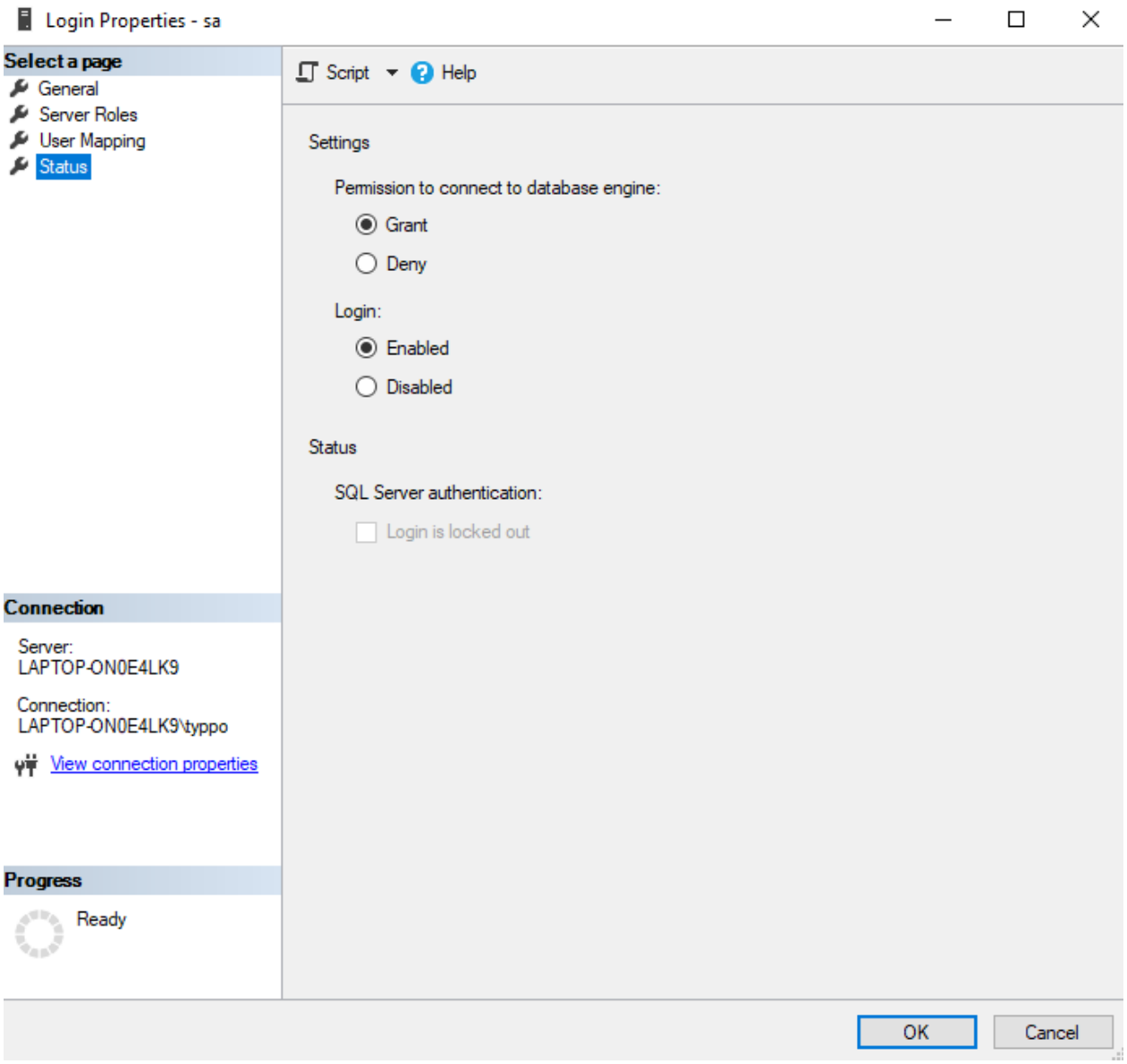
Connection:
LAPTOP-ON0E4LK9\typpo

View connection properties

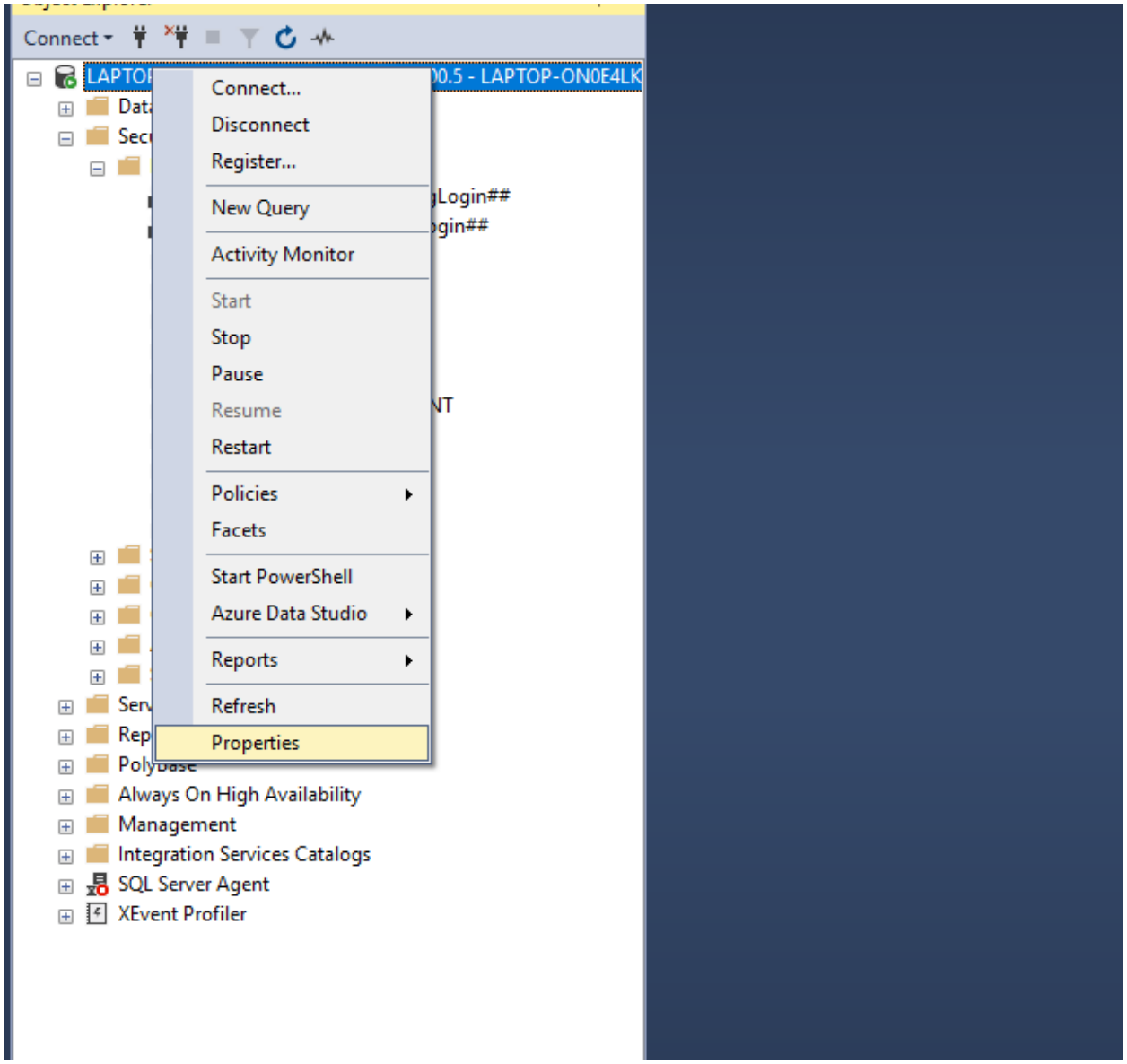
Progress

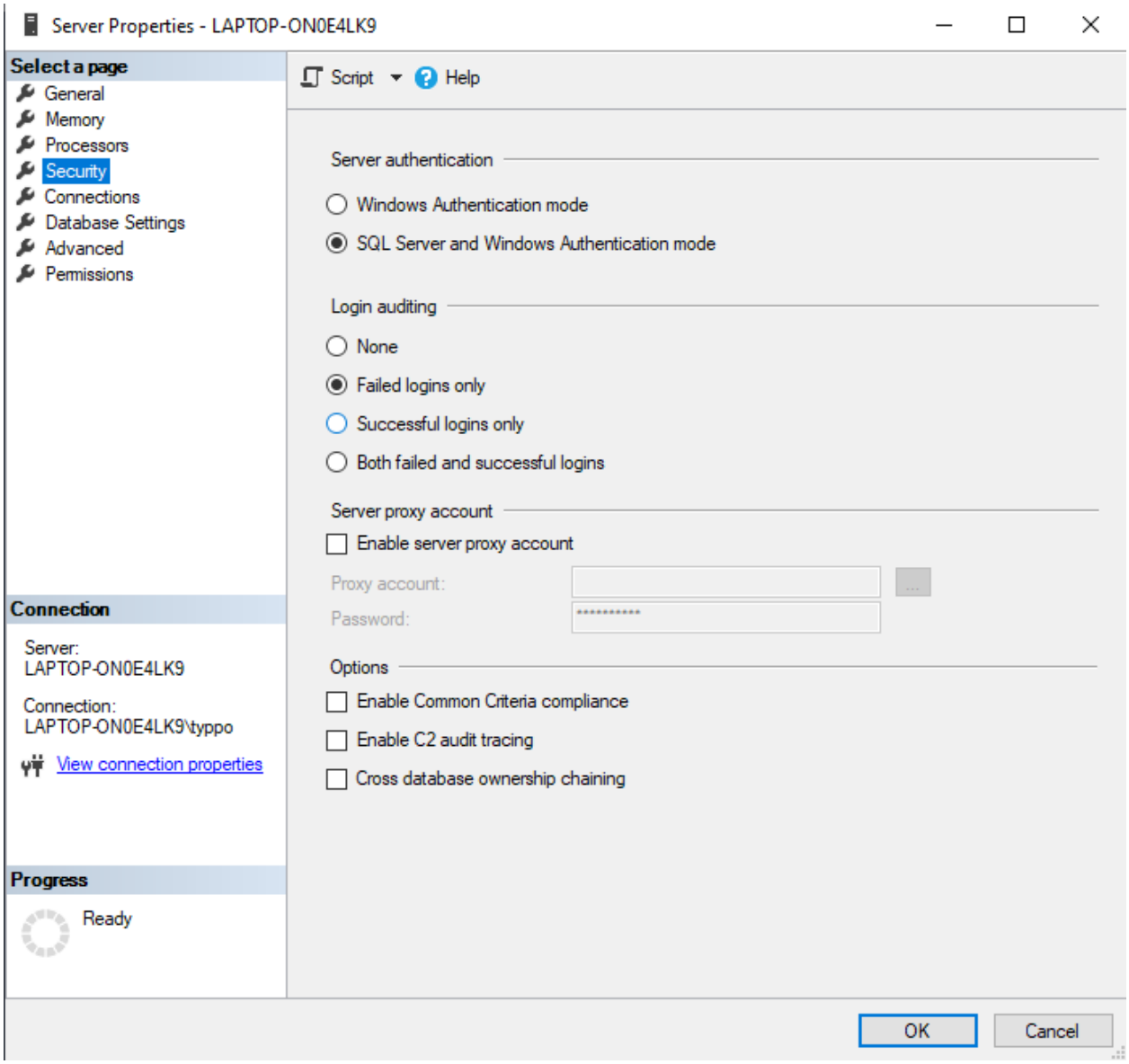
Ready

OK Cancel



Remember to set the server properties like below


Connect
Connect...
Disconnect
Register...
New Query
Activity Monitor
Start
Stop
Pause
Resume
Restart
Policies
Facets
Start PowerShell
Azure Data Studio
Reports
Refresh
Properties
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent
XEvent Profiler



Get your ip info from “ip config” command

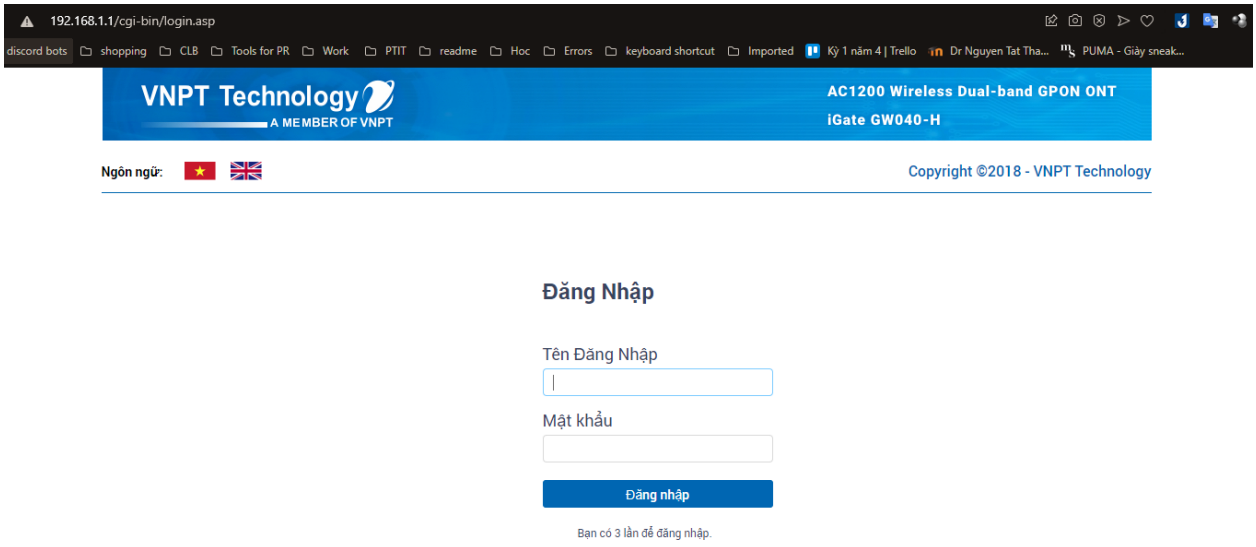
Press Window + R -> type “cmd” -> OK, type “ipconfig” -> OK

```
C:\Users\typpo>ipconfig

Windows IP Configuration


Ethernet adapter Radmin VPN:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : fdfd::1ad0:ceac
   Link-local IPv6 Address . . . . . : fe80::953c:ef78:8600:22a7%22
   IPv4 Address. . . . . . . . . . . : 26.208.206.172
   Subnet Mask . . . . . . . . . . . : 255.0.0.0
   Default Gateway . . . . . . . . . : 26.0.0.1

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::c173:82ff:3dbf:75fd%14
   IPv4 Address. . . . . . . . . . . : 192.168.1.10
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 11:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 12:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:2851:fcb0:20ce:1ad0:84ee:7852
   Link-local IPv6 Address . . . . . : fe80::20ce:1ad0:84ee:7852%16
   Default Gateway . . . . . . . . . :

C:\Users\typpo>
```

Keep in mind the “IPv4 Address” and the “Default Gateway”

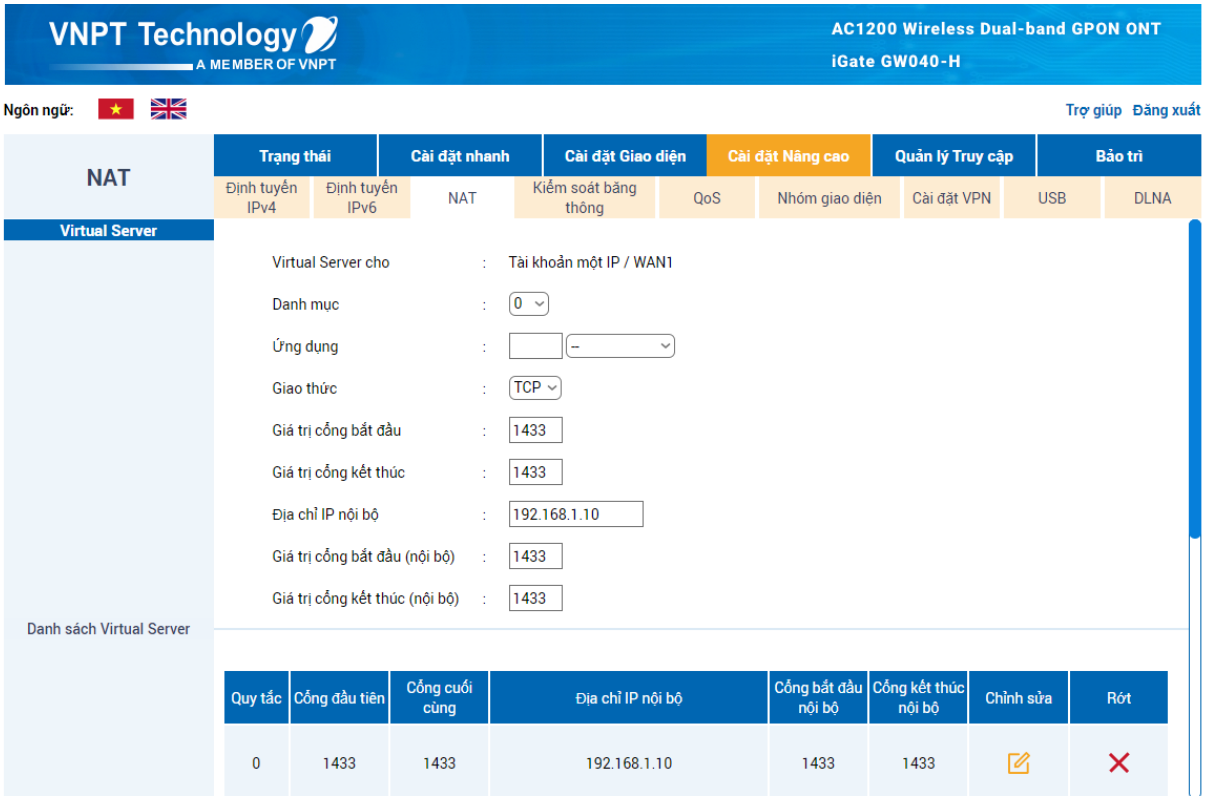
Port forwarding to open port on router

Go to your “Default Gateway” using browser

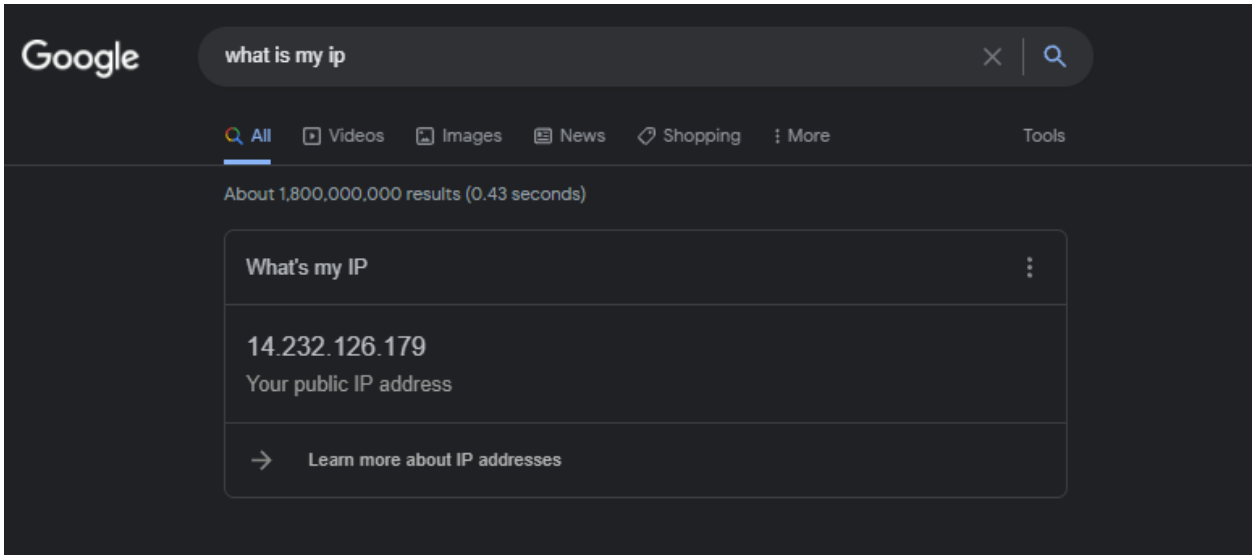


Find “advanced setting” page and it depends on the router

Open the port “1433” like below

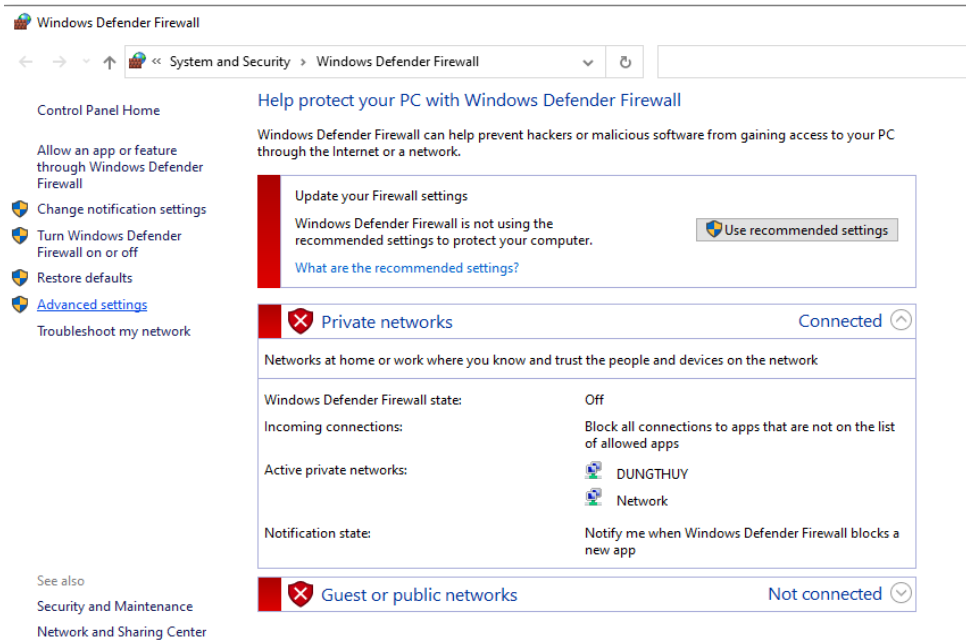


Find public ip using “what is my ip” to connect later

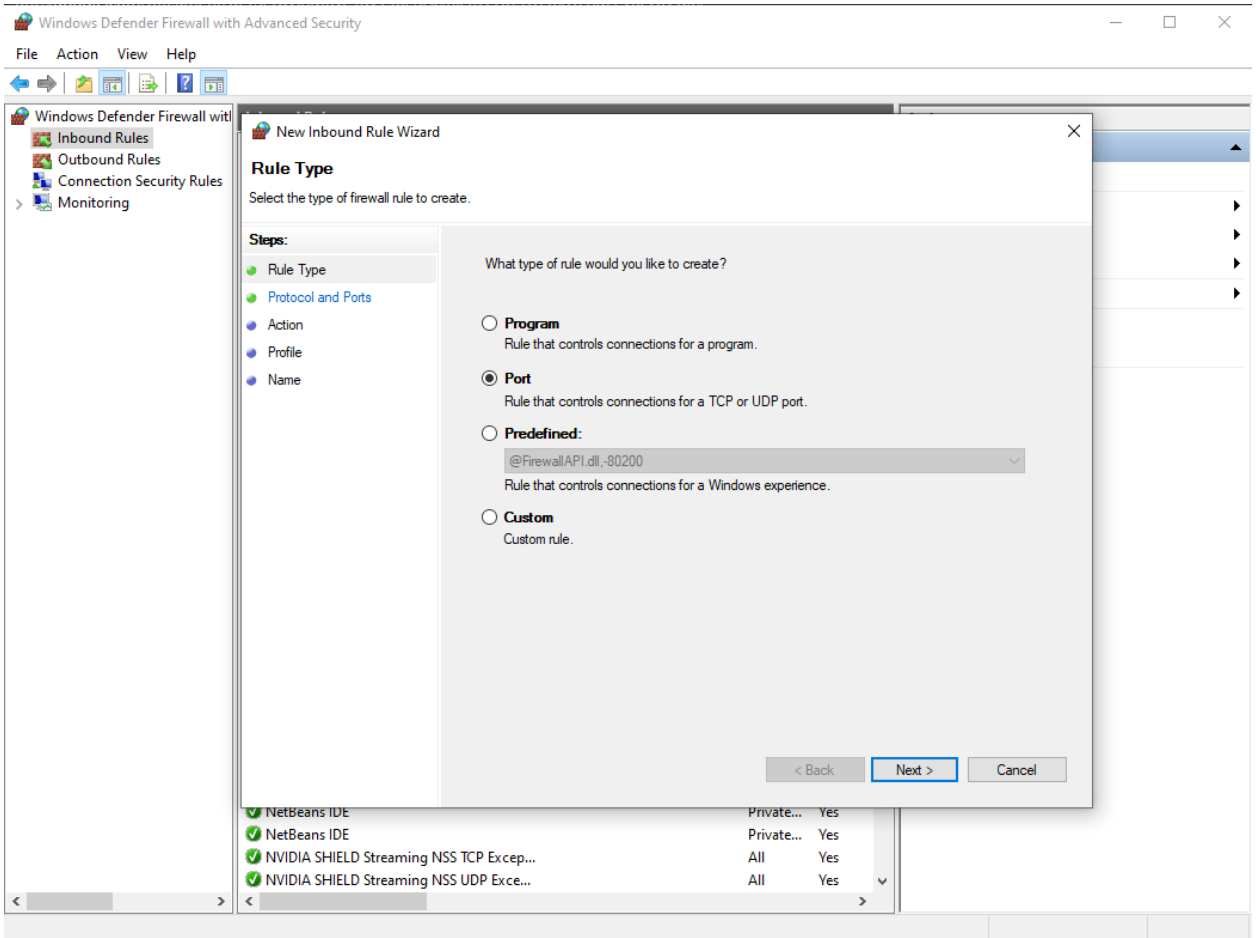


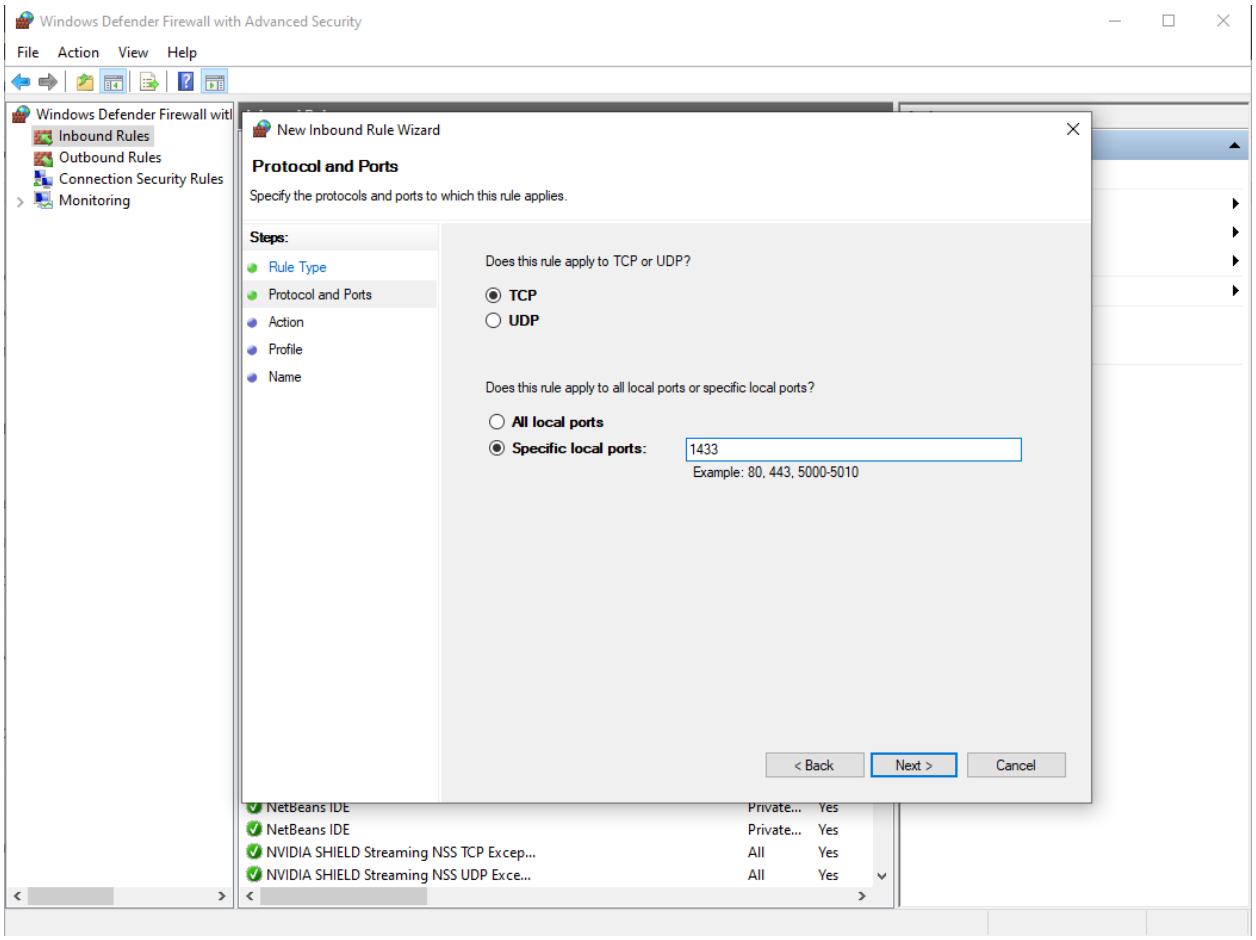
Open port “1433” via setting inbound, outbound rules

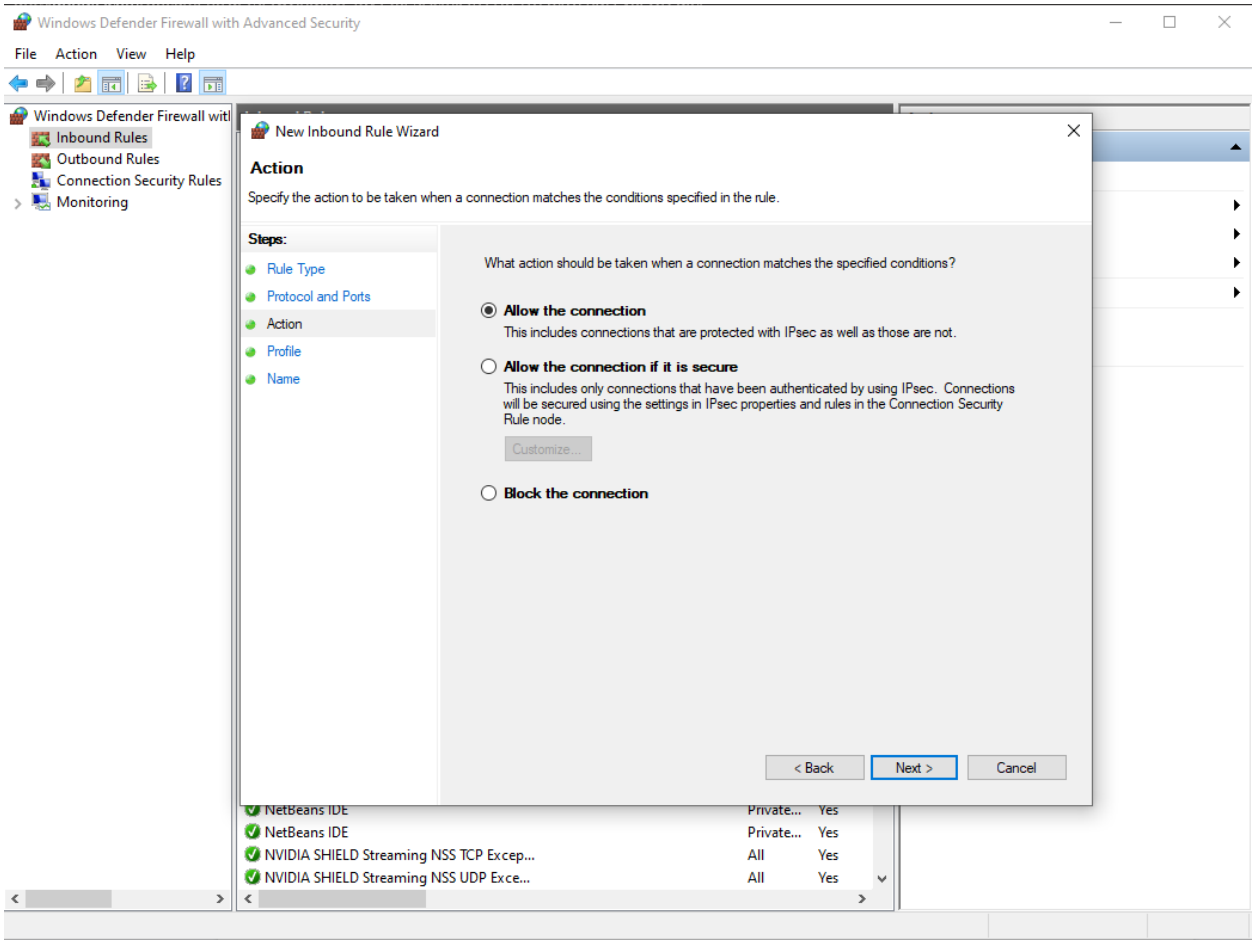
Go to “Windows Firewall” -> “Advanced settings”

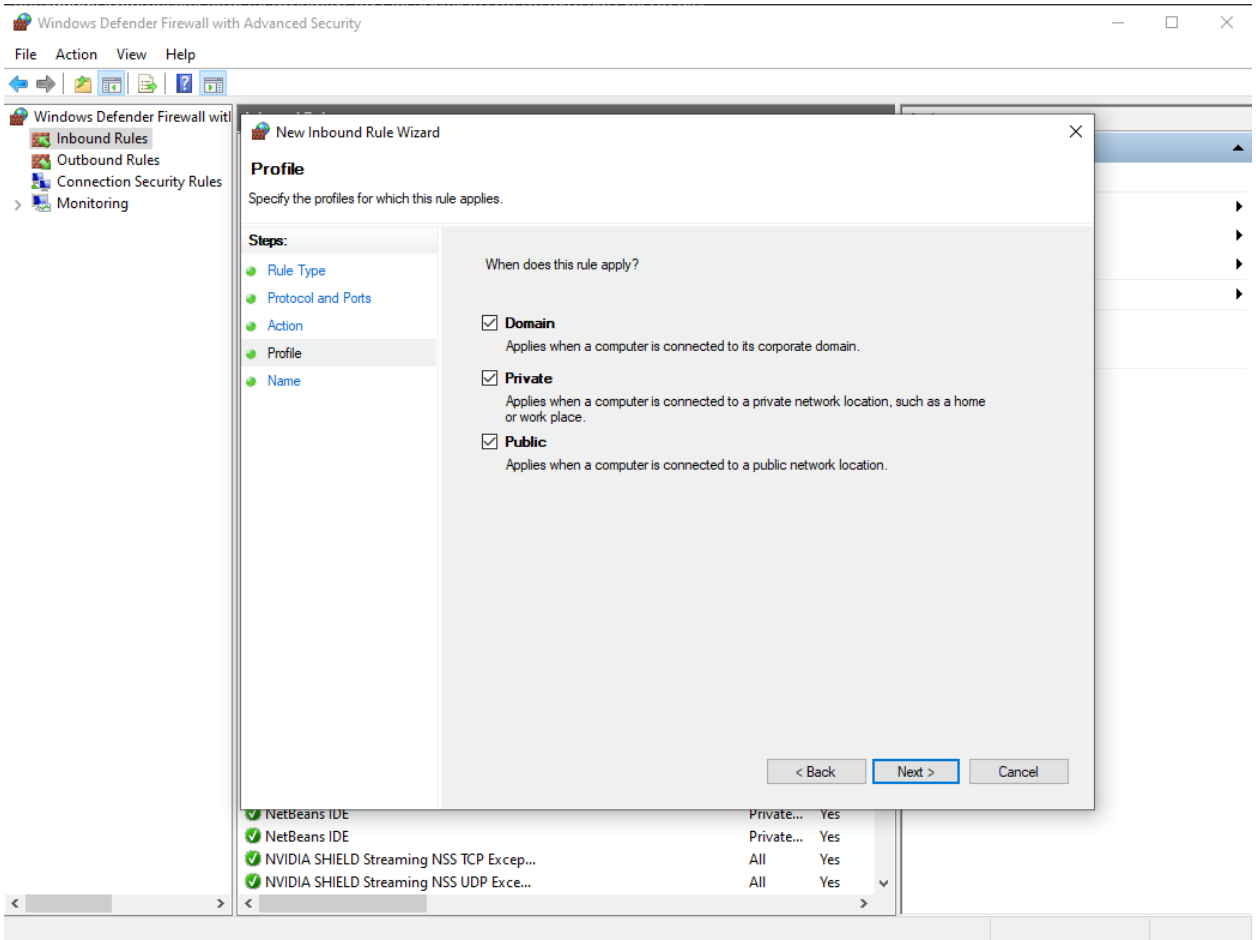


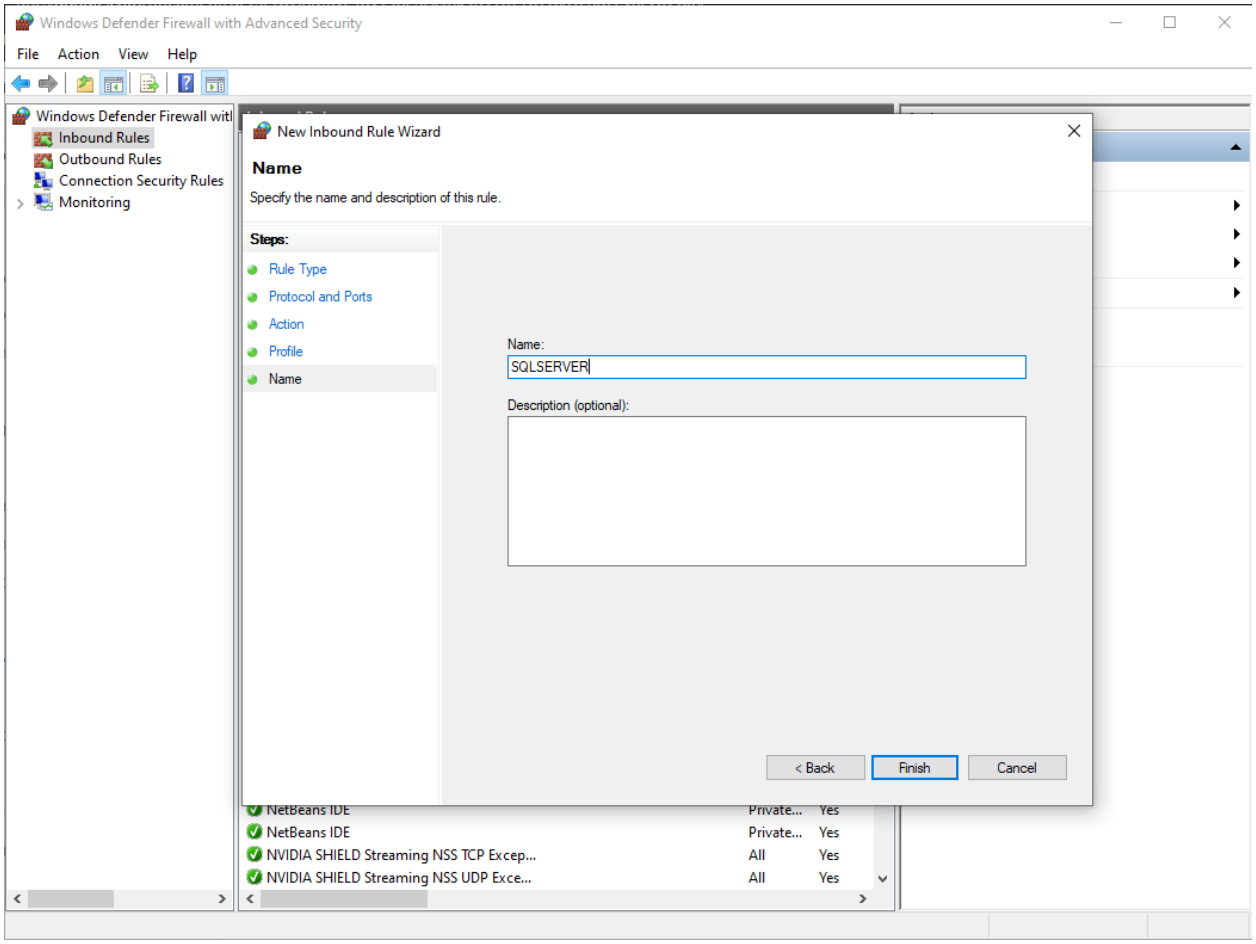
Go to “Inbound/Outbound rules” -> “New rules”, then setup like below



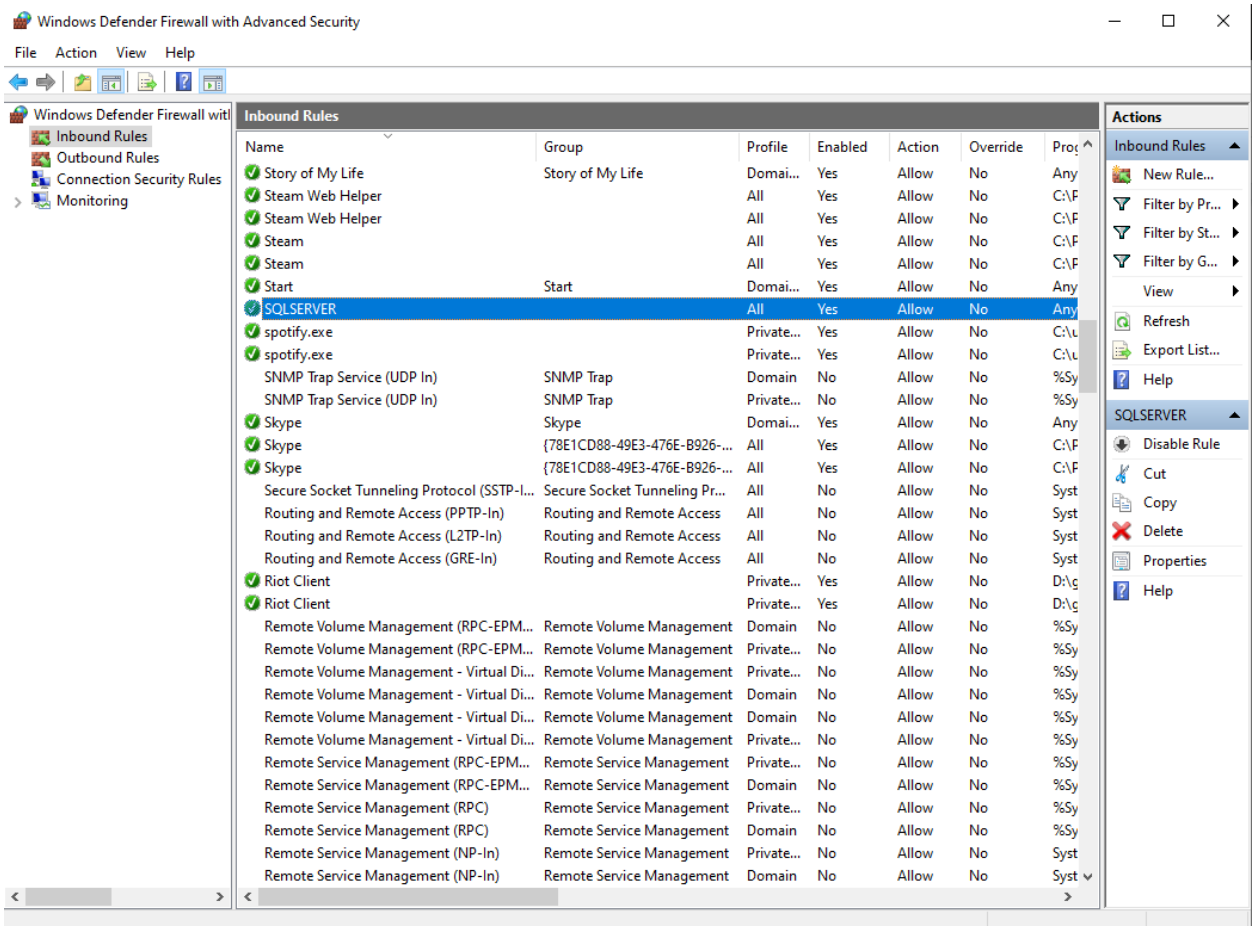

Windows Defender Firewall with Advanced Security
File Action View Help
Windows Defender Firewall witl
Inbound Rules
Outbound Rules
Connection Security Rules
Monitoring
New Inbound Rule Wizard
Protocol and Ports
Specify the protocols and ports to which this rule applies.
Steps:
Rule Type
Protocol and Ports
Action
Profile
Name
Does this rule apply to TCP or UDP?
TCP
UDP
Does this rule apply to all local ports or specific local ports?
All local ports
Specific local ports:
1433
Example: 80, 443, 5000-5010
< Back
Next >
Cancel
NetBeans IDE
NetBeans IDE
NVIDIA SHIELD Streaming NSS TCP Excep...
NVIDIA SHIELD Streaming NSS UDP Exce...


Windows Defender Firewall with Advanced Security
File Action View Help
Windows Defender Firewall with
Inbound Rules
Outbound Rules
Connection Security Rules
Monitoring
New Inbound Rule Wizard
Action
Specify the action to be taken when a connection matches the conditions specified in the rule.
Steps:
Rule Type
Protocol and Ports
Action
Profile
Name
What action should be taken when a connection matches the specified conditions?
Allow the connection
This includes connections that are protected with IPsec as well as those are not.
Allow the connection if it is secure
This includes only connections that have been authenticated by using IPsec. Connections will be secured using the settings in IPsec properties and rules in the Connection Security Rule node.
Customize...
Block the connection
< Back
Next >
Cancel
NetBeans IDE Private... Yes
NVIDIA SHIELD Streaming NSS TCP Excep... All Yes
NVIDIA SHIELD Streaming NSS UDP Exce... All Yes


Windows Defender Firewall with Advanced Security
File Action View Help
Windows Defender Firewall with
Inbound Rules
Outbound Rules
Connection Security Rules
Monitoring
New Inbound Rule Wizard
Profile
Specify the profiles for which this rule applies.
Steps:
Rule Type
Protocol and Ports
Action
Profile
Name
When does this rule apply?
Domain
Applies when a computer is connected to its corporate domain.
Private
Applies when a computer is connected to a private network location, such as a home or work place.
Public
Applies when a computer is connected to a public network location.
< Back
Next >
Cancel
NetBeans IDE Private... Yes
NVIDIA SHIELD Streaming NSS TCP Excep... All Yes
NVIDIA SHIELD Streaming NSS UDP Exce... All Yes



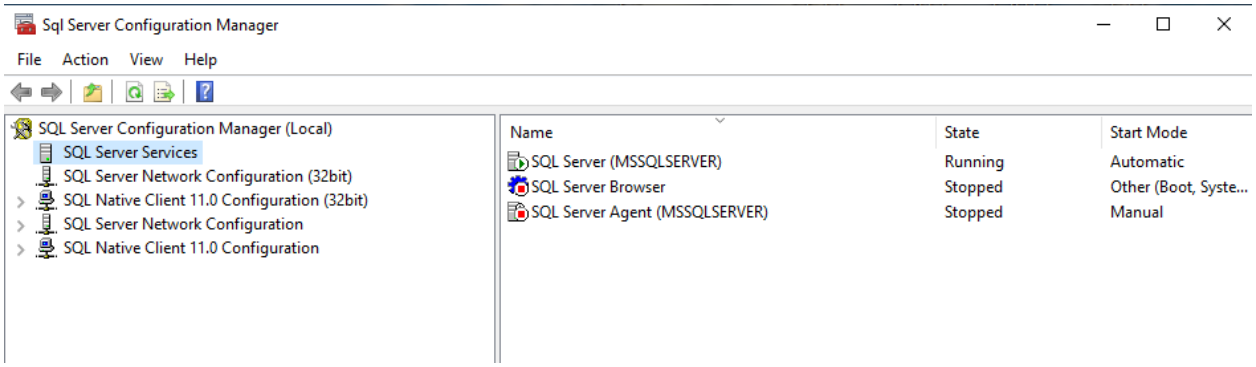
The result may look like this:



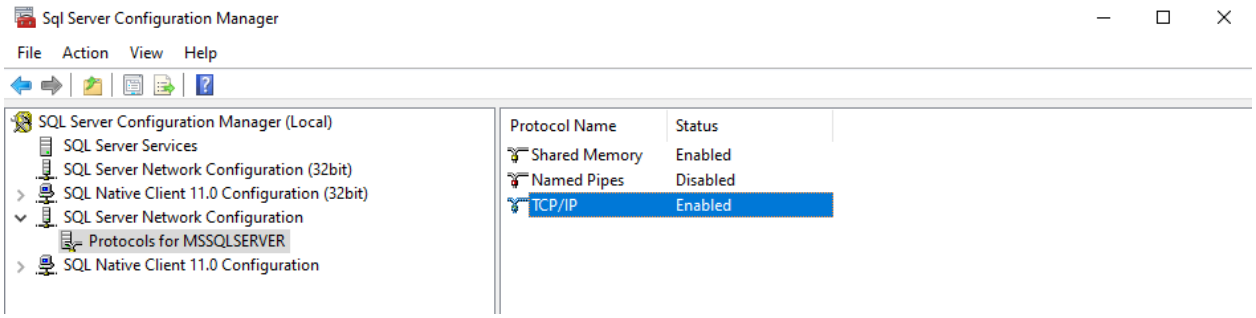
Remember to set the “outbound rules” correspondingly

Enable TCP protocols using SQL Server Configuration Manager

Open the “SQL Server configuration manager”

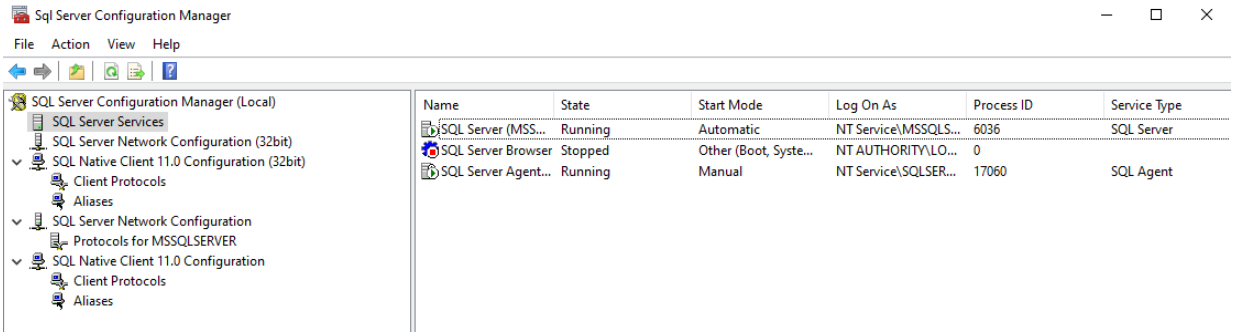


Set the TCP/IP protocols for your SQLServer to “Enabled”



Set all the “Client protocols”’s subcategory’s TCP/IP protocols to “Enabled”

When running your server, make sure all the required services are running like below

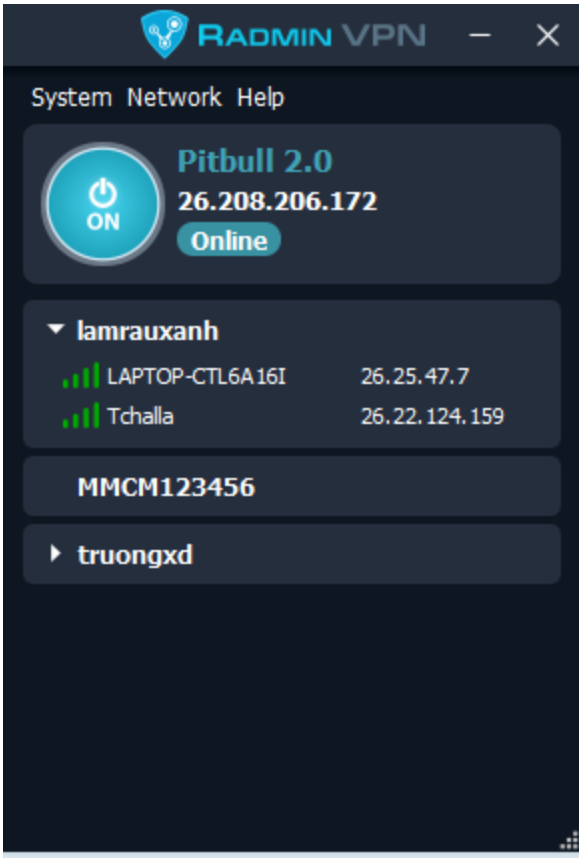


Use query to create new tables, insert records to tables

Create a new database in your SQLServer, and insert records to it.

Set up Virtual LAN network for our servers using VPN (tool: RadminVPN)

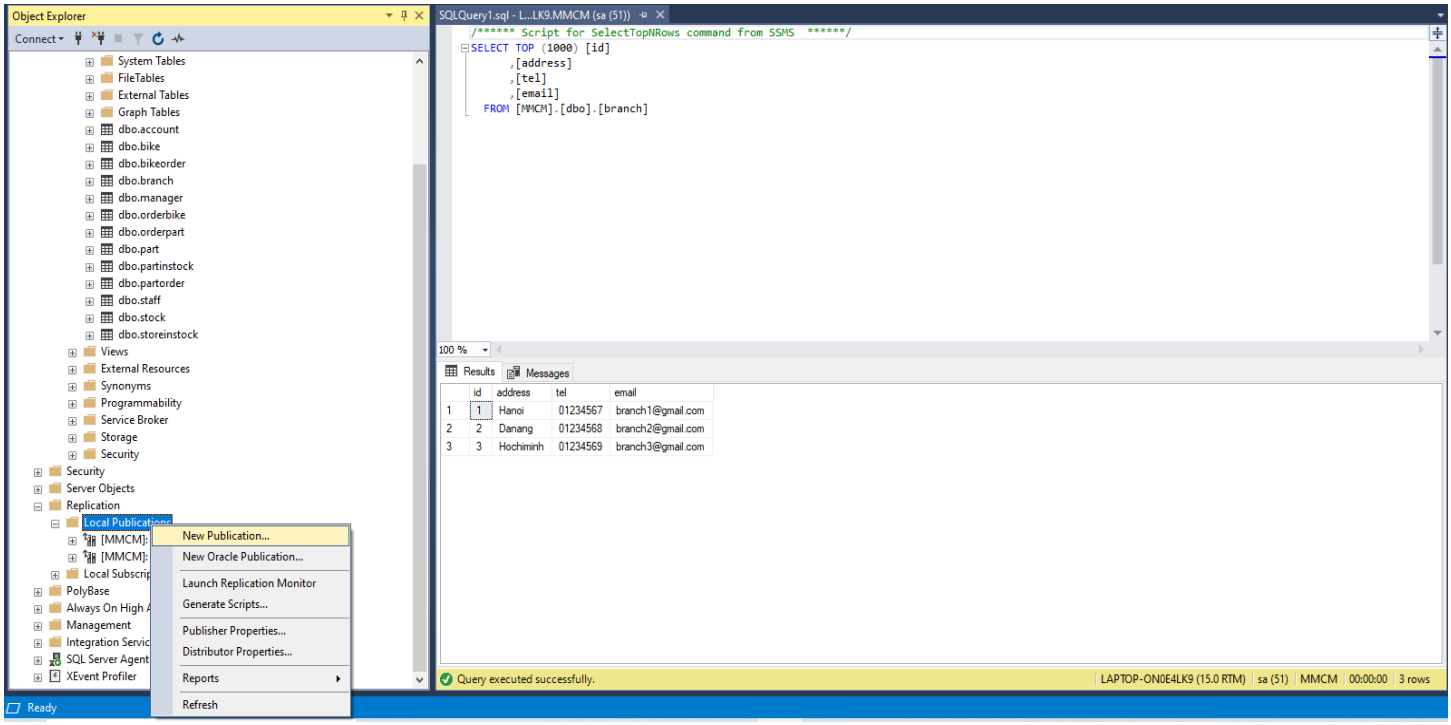
Create a network and then all the machines join into it



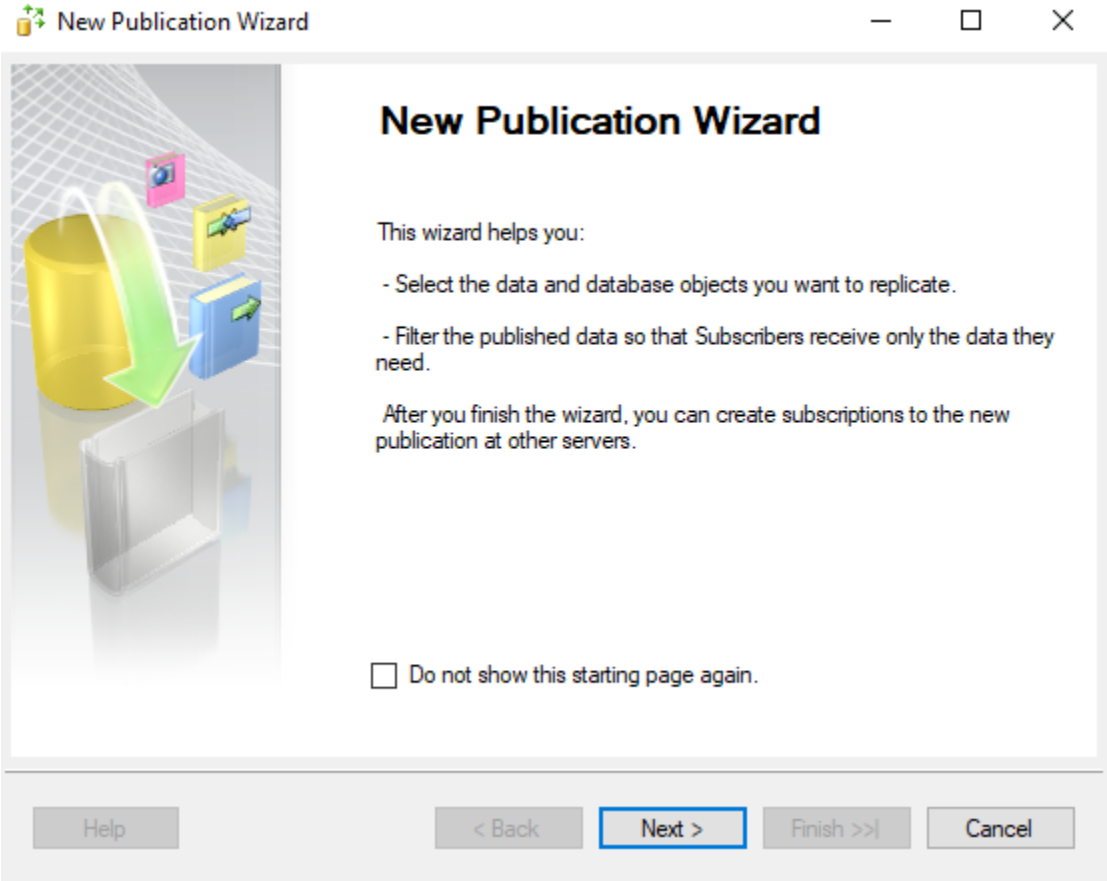
VPN provides reliability and stability to our system.

Add Publication

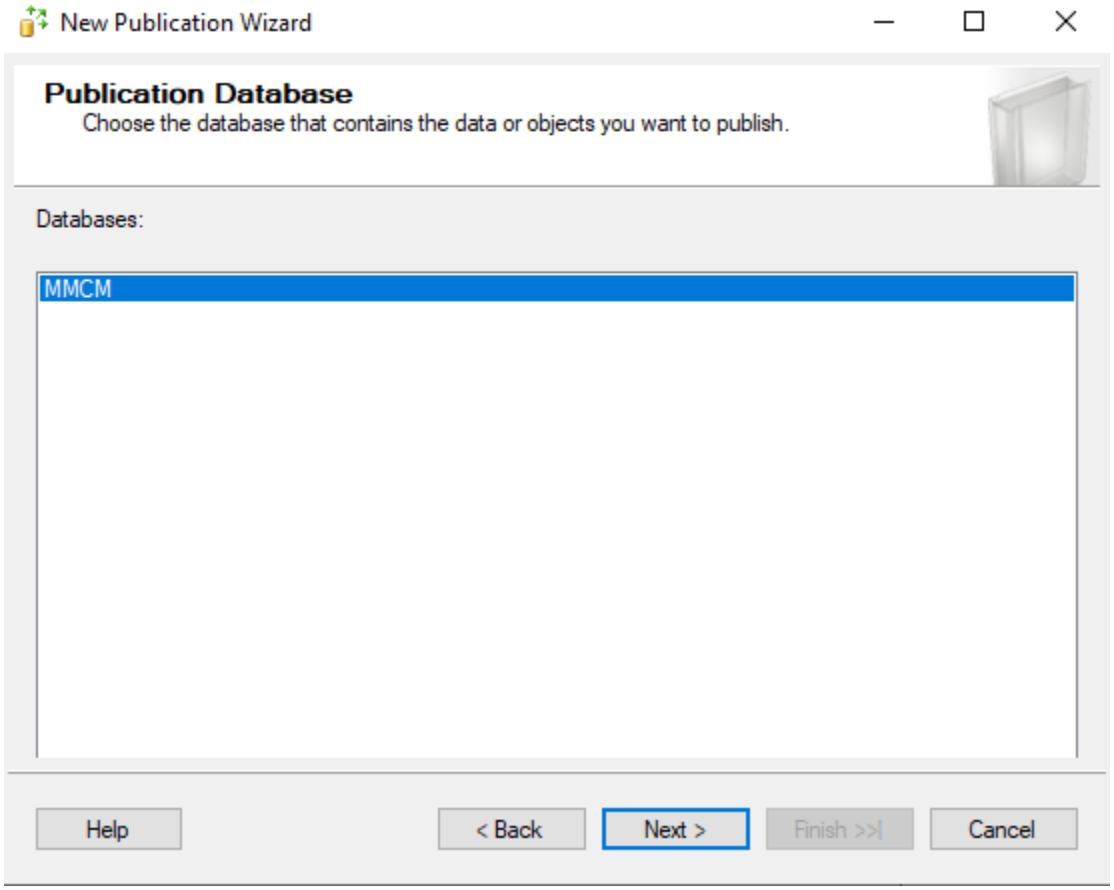
After finishing creating the initial database, we start distributing. First, in the database folder, double click “Replication”, then inside that right click on the “Local publications” folder and choose “New Publication...”



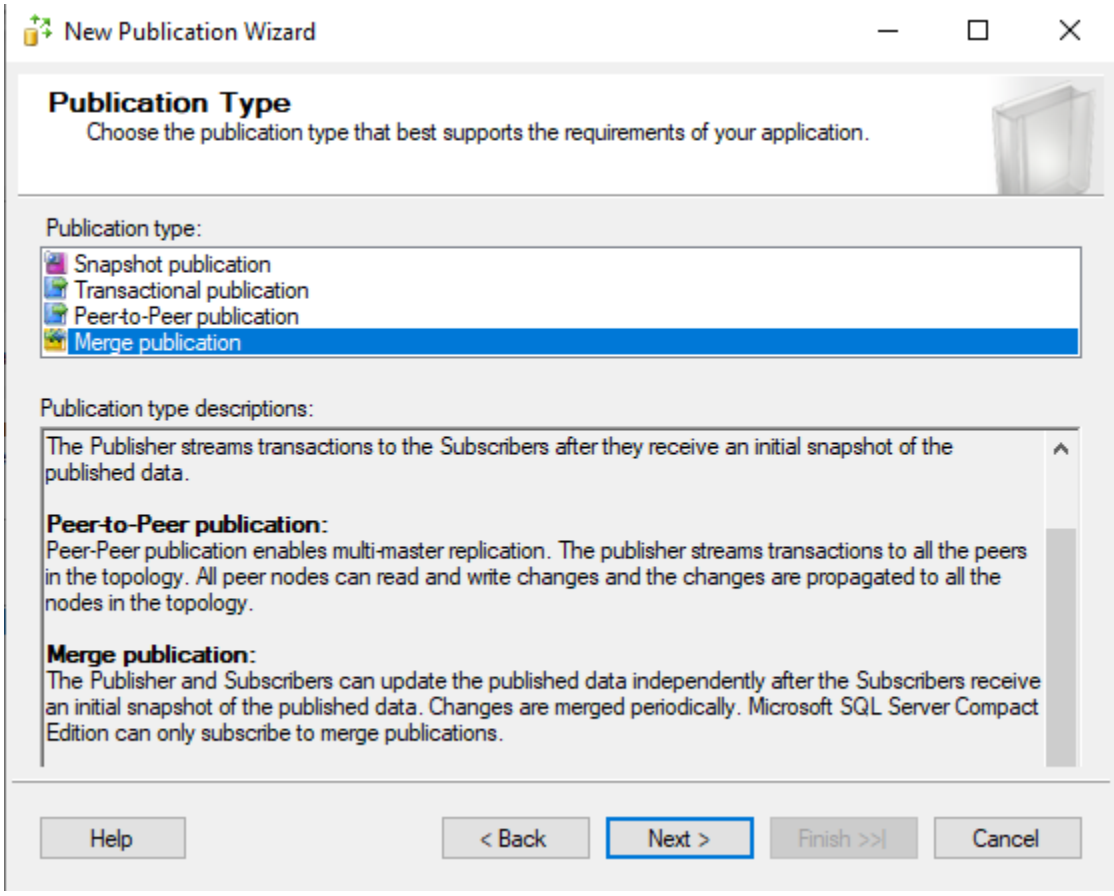
The publication wizard appears, click next



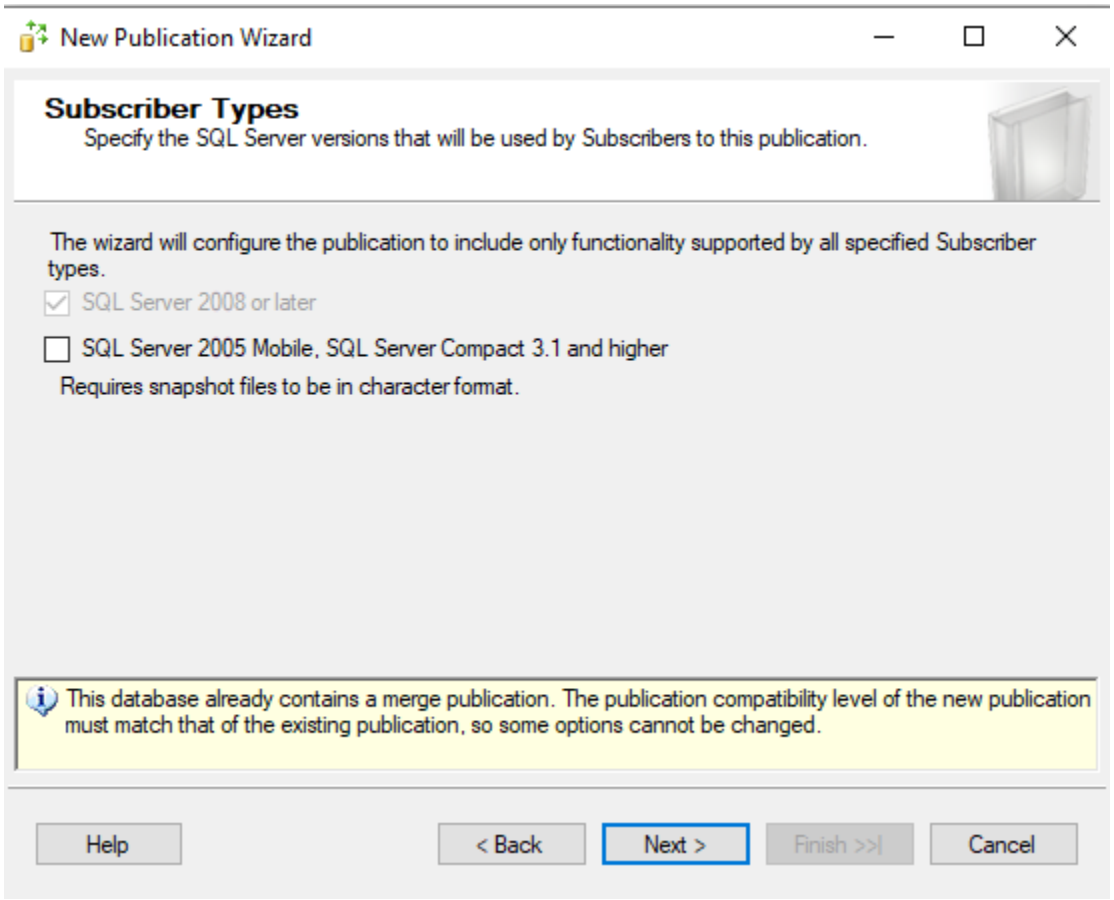
Choose the desired database to create publications, in this case “MMCM”, Then click “next”



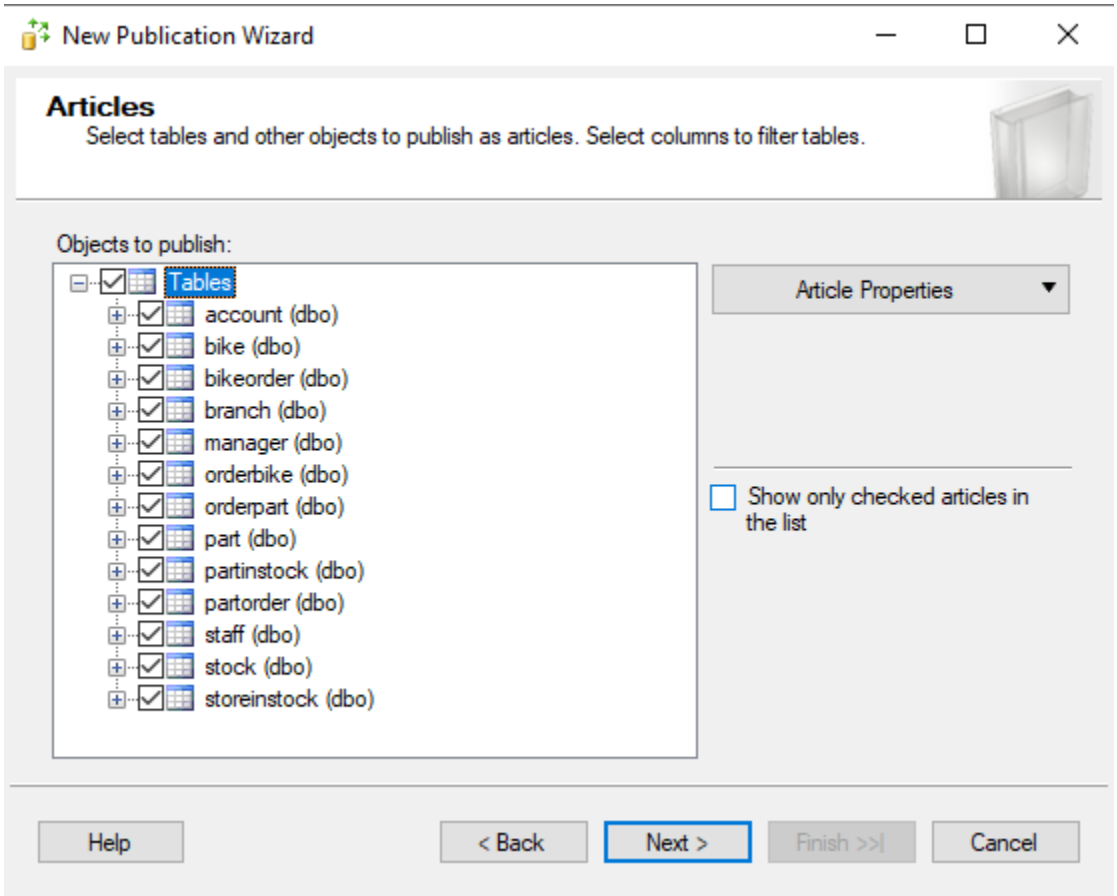
Select the “Merge publication” option



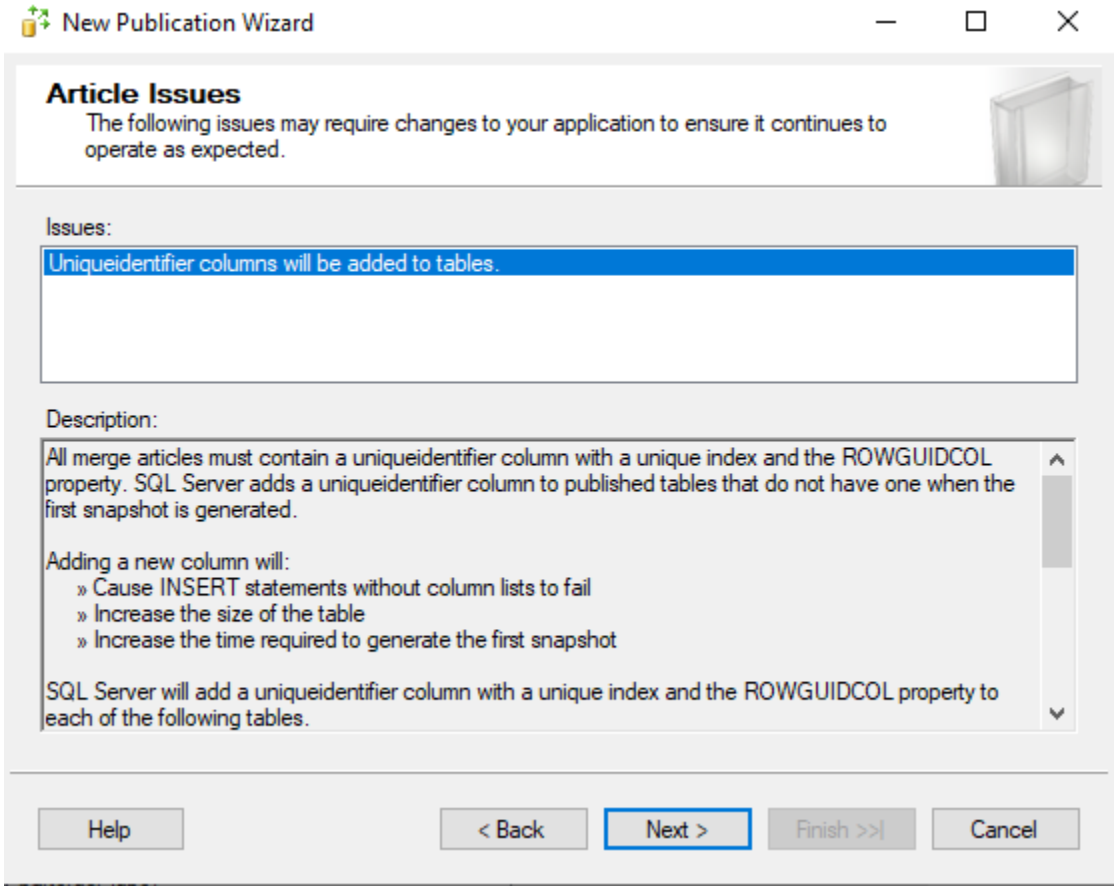
Select subscriber types, then click “next”



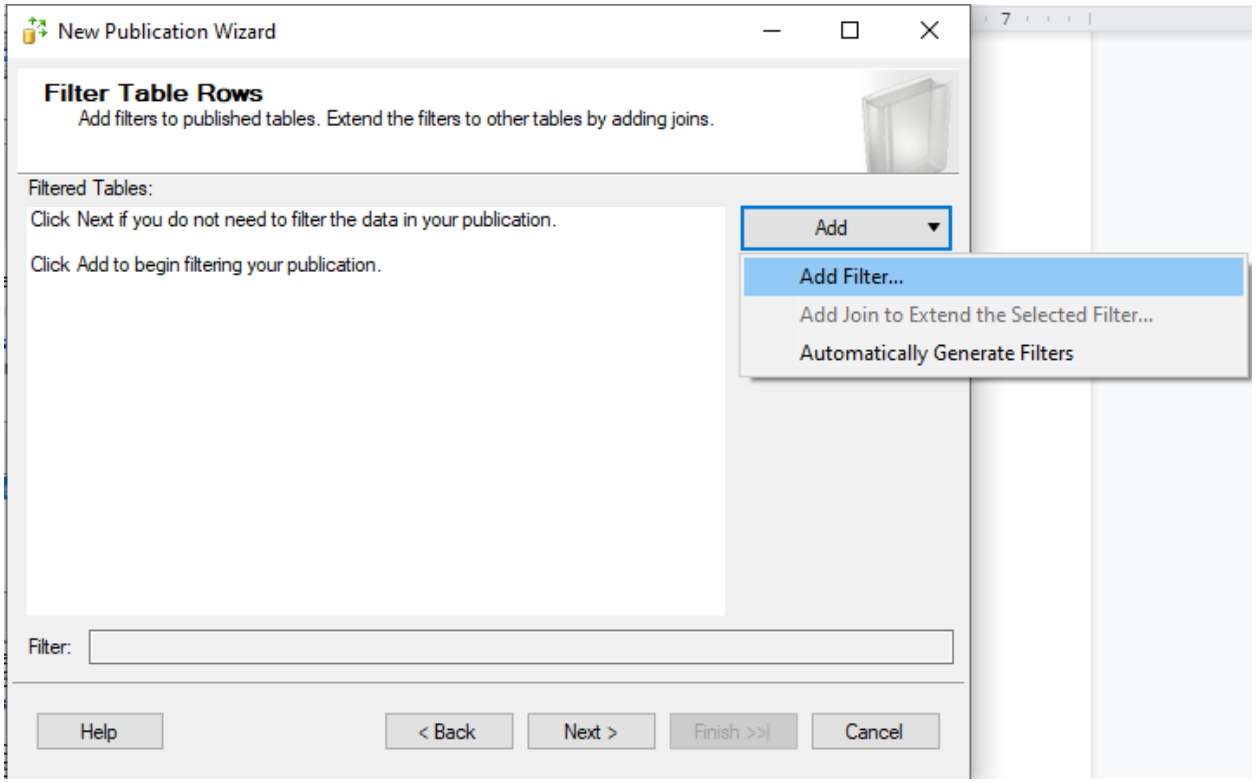
Select tables in database to create publication

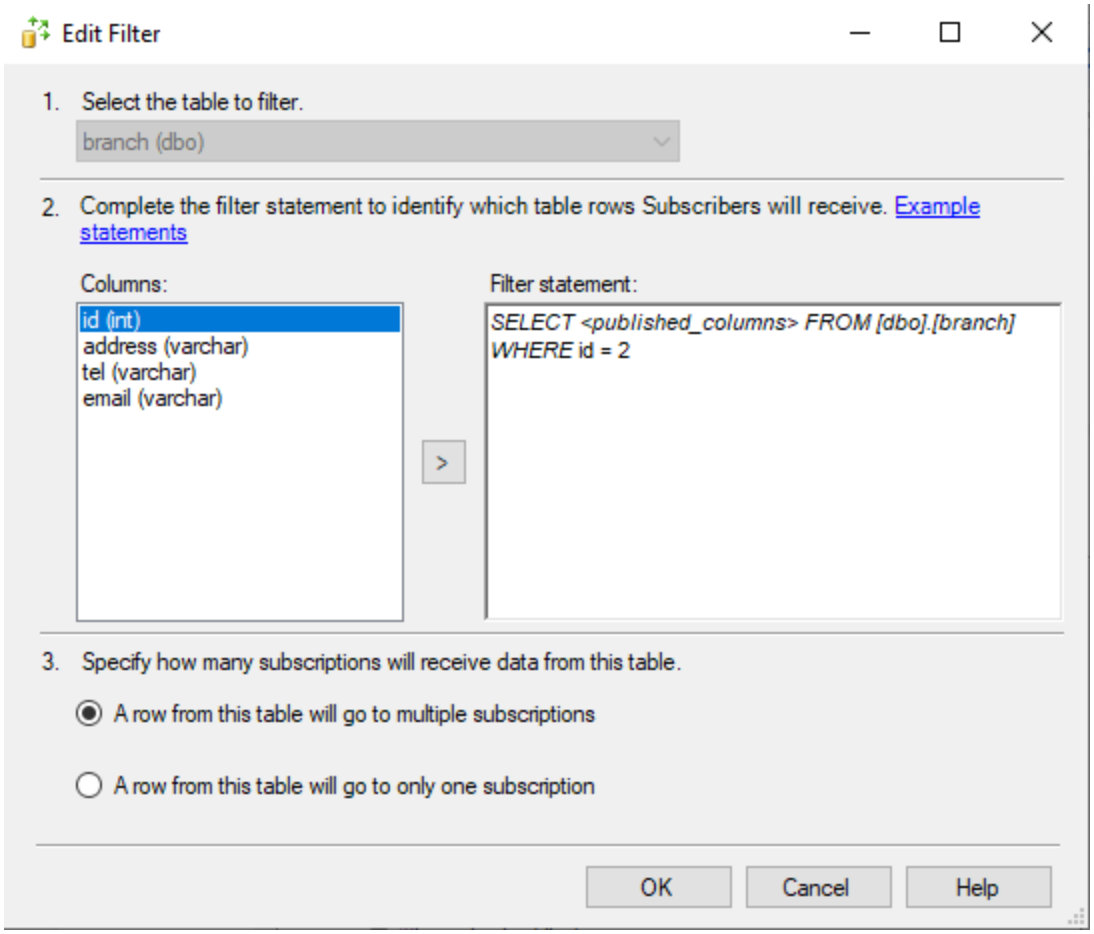


This page is to remind users that a new column(Uniqueidentifier) will be added to tables. Click “next”.

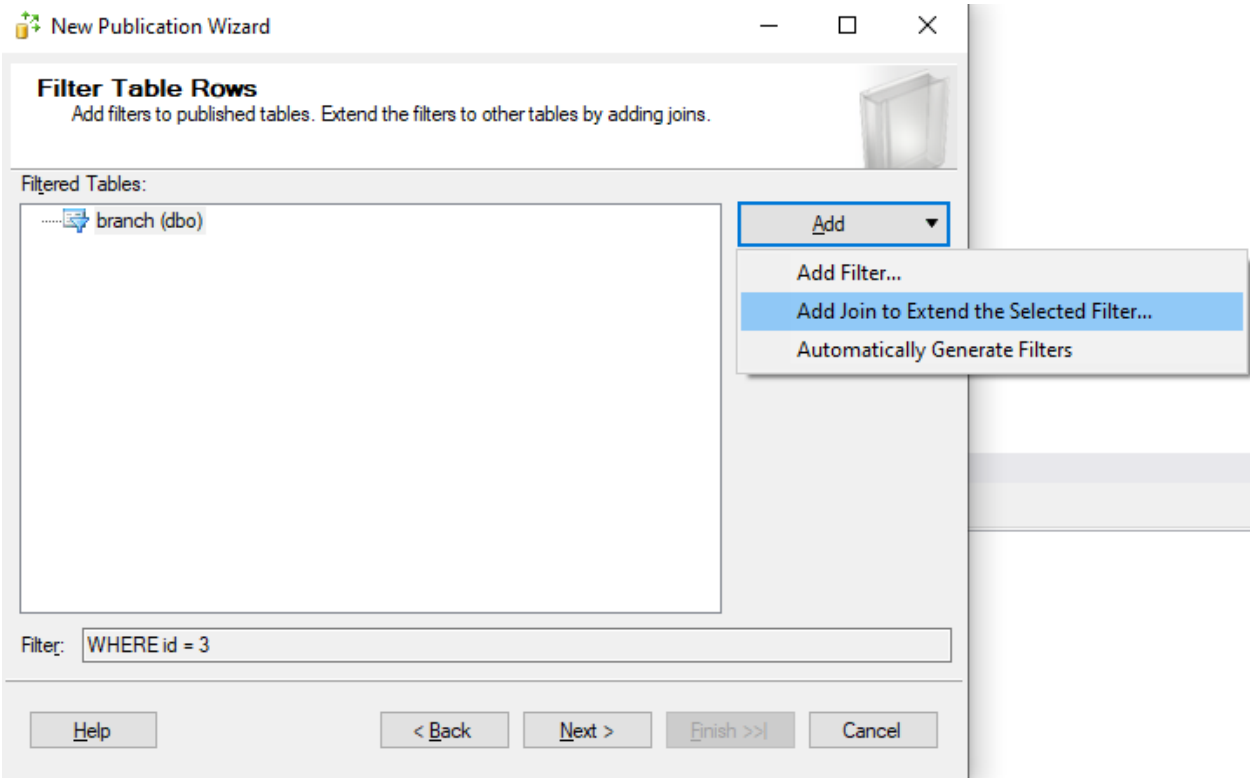


Add filters, which is the same as the minterms that are used to fragmenting data in the data fragmenting part

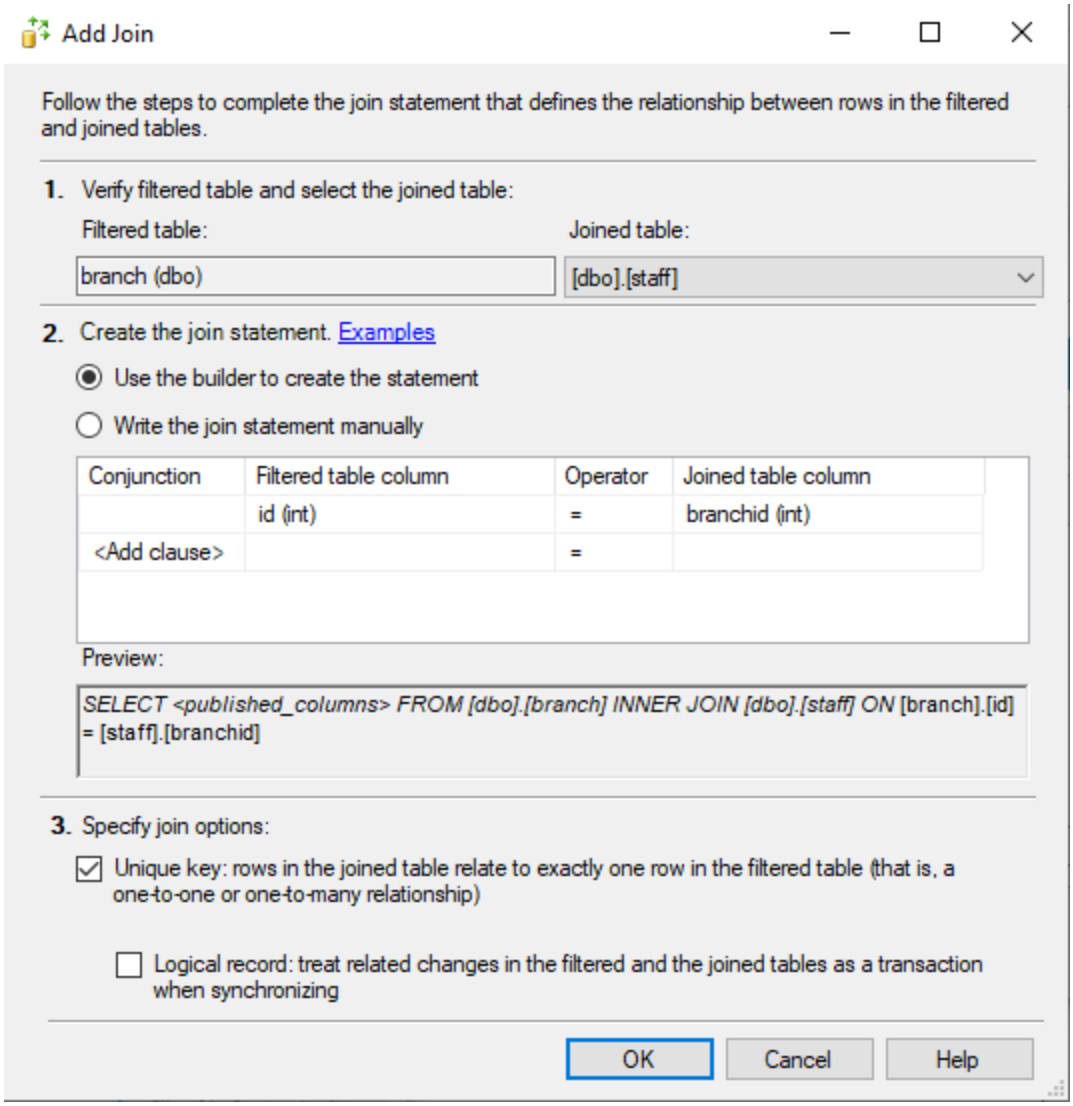




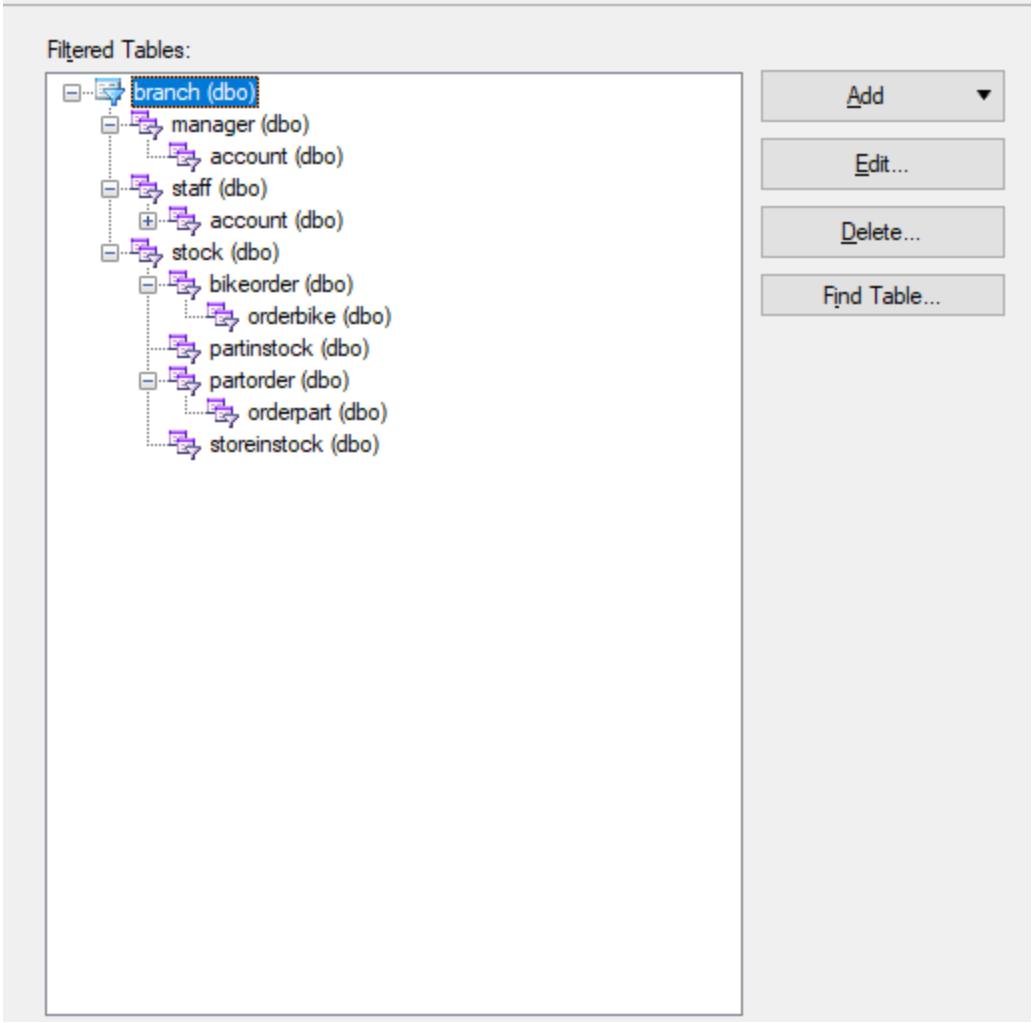
Add join for the above filter:



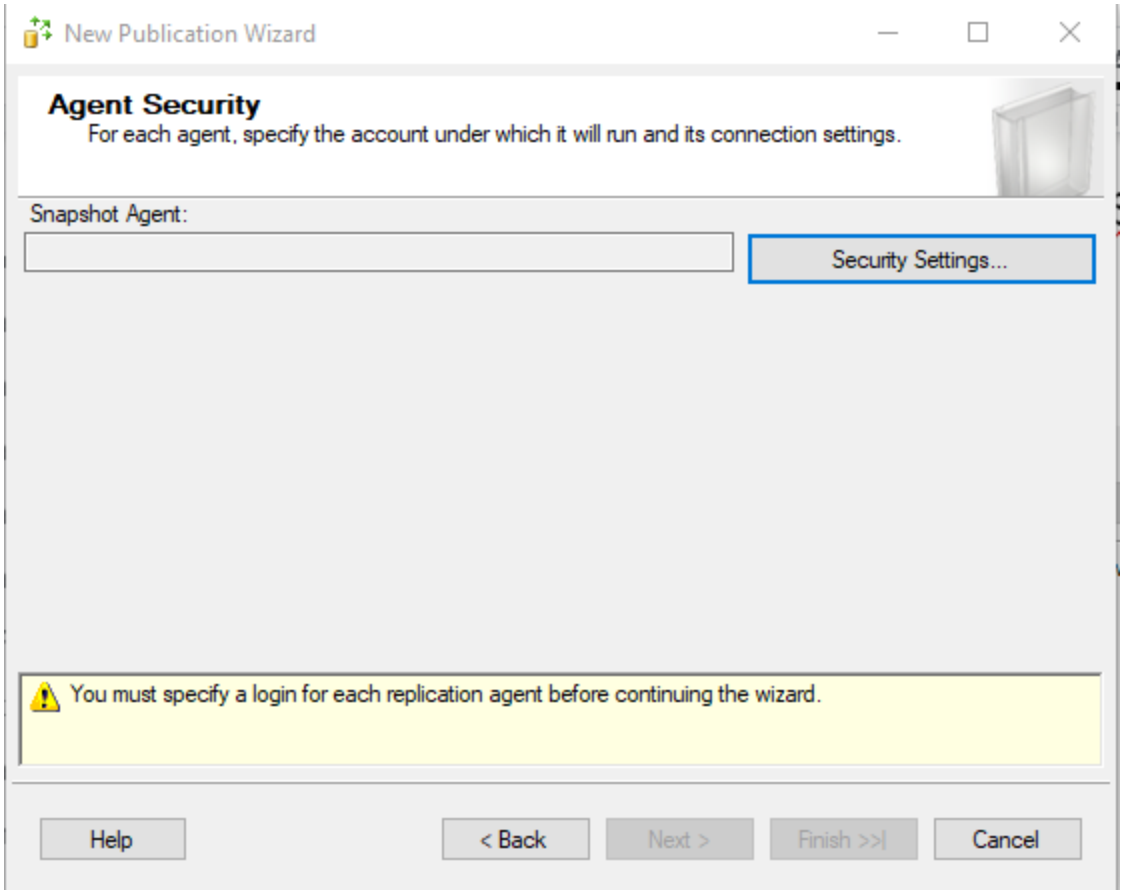
Choose the corresponding columns



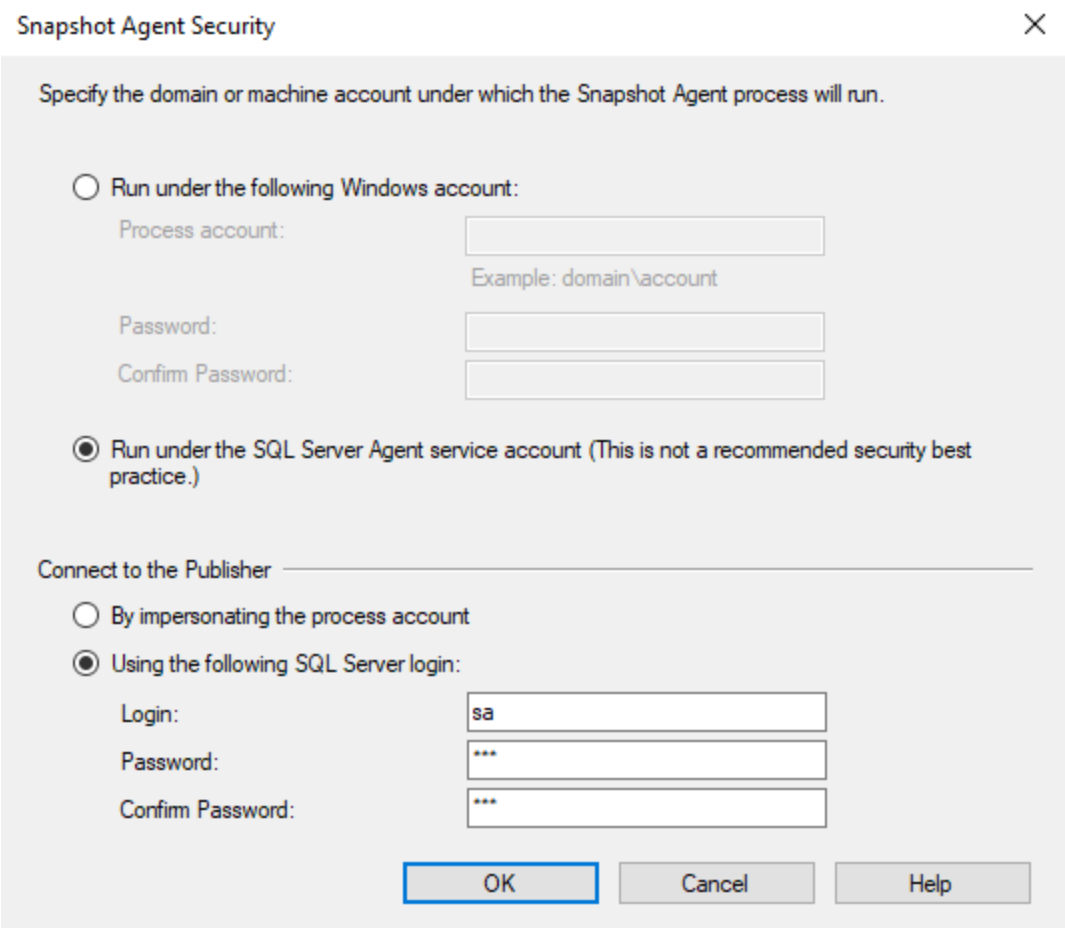
The resulting filter may look like this (for the implementation, we decide to not fragmenting the “bike” and “part” table, despite of the previous horizontal fragmentation above)


Filtered Tables:
branch (dbo)
manager (dbo)
account (dbo)
staff (dbo)
account (dbo)
stock (dbo)
bikeorder (dbo)
orderbike (dbo)
partinstock (dbo)
partorder (dbo)
orderpart (dbo)
storeinstock (dbo)
Add
Edit...
Delete...
Find Table...

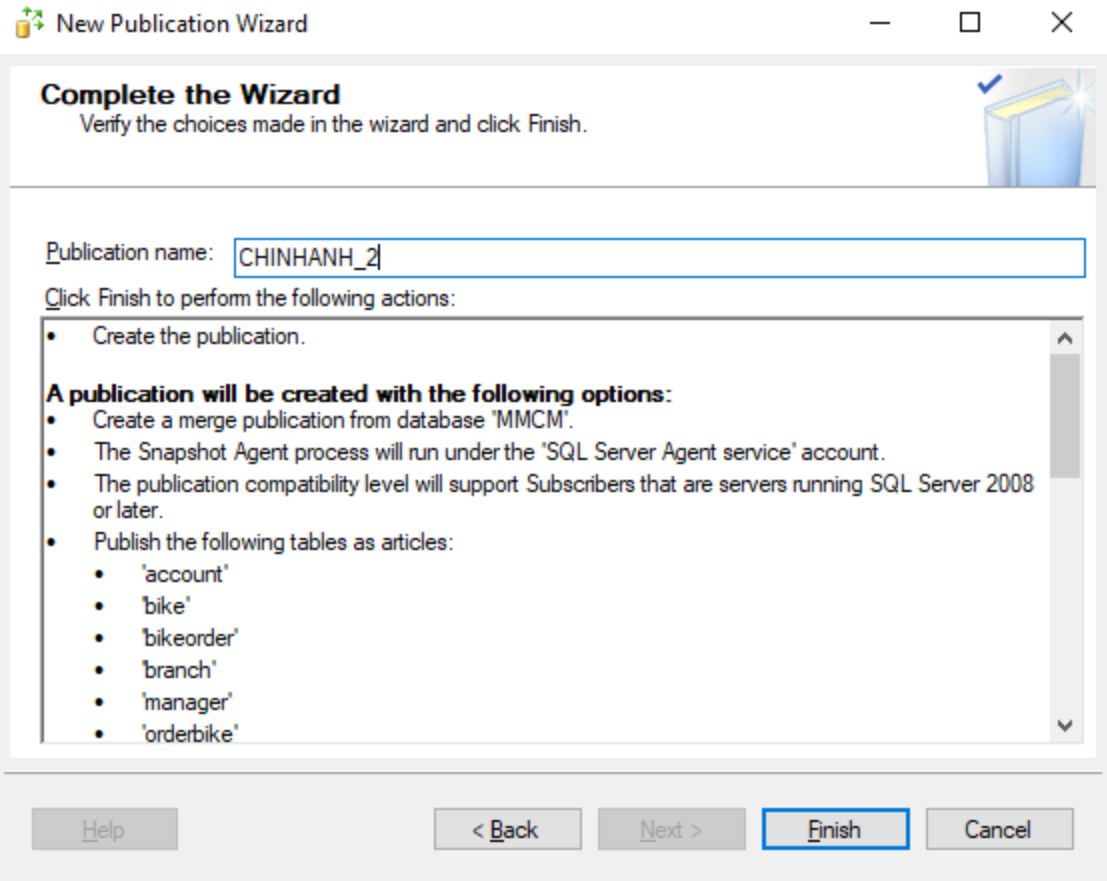
Click “Next”



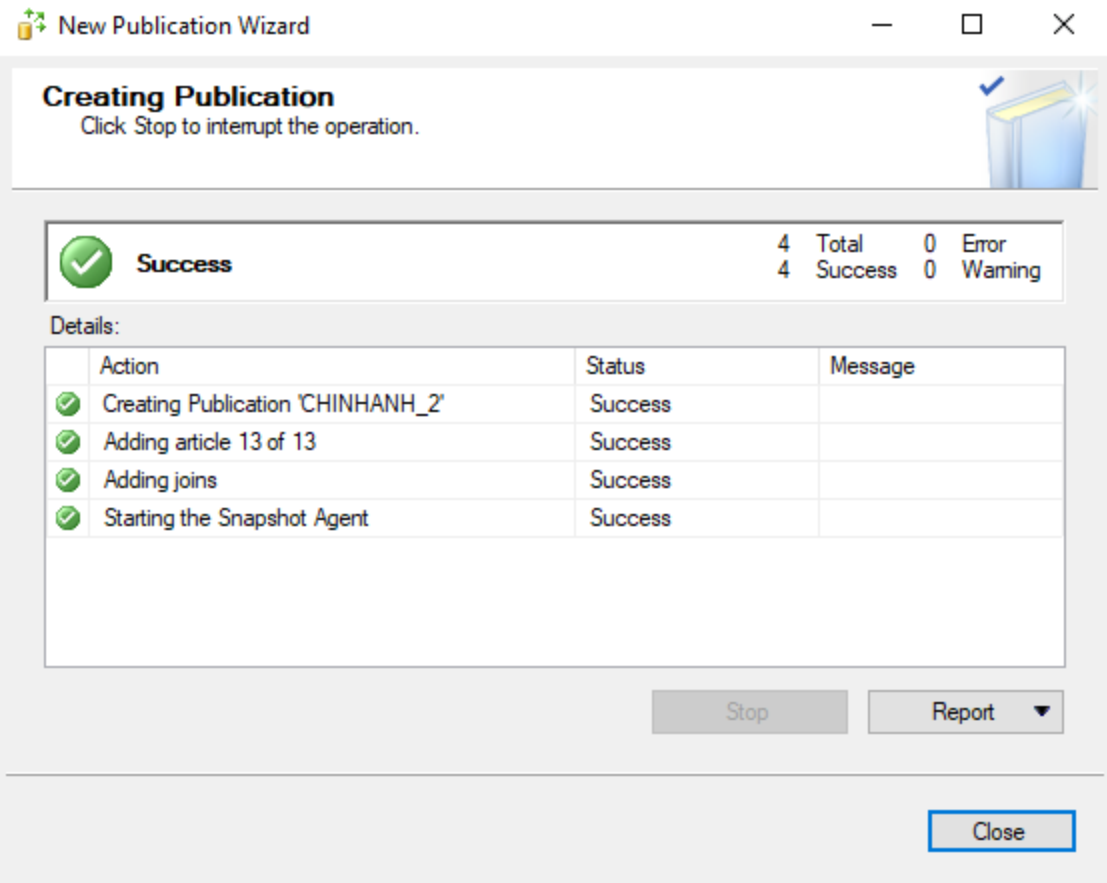
Click “Security Settings” -> choose “Run under the SQL Server Agent service account” and “Using the following SQL Server login” -> enter user “sa”



Click “OK”, enter publication name -> “finish”

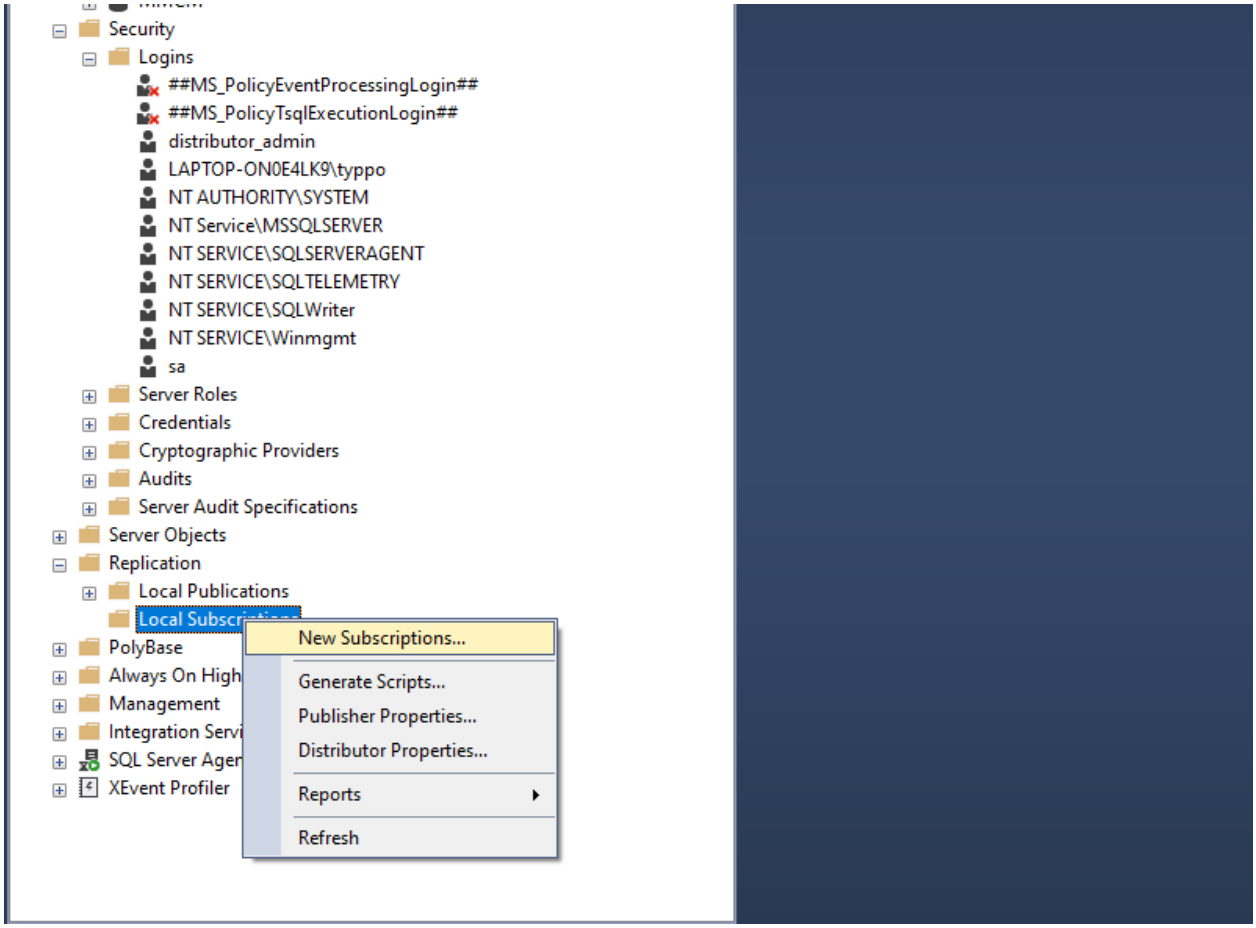


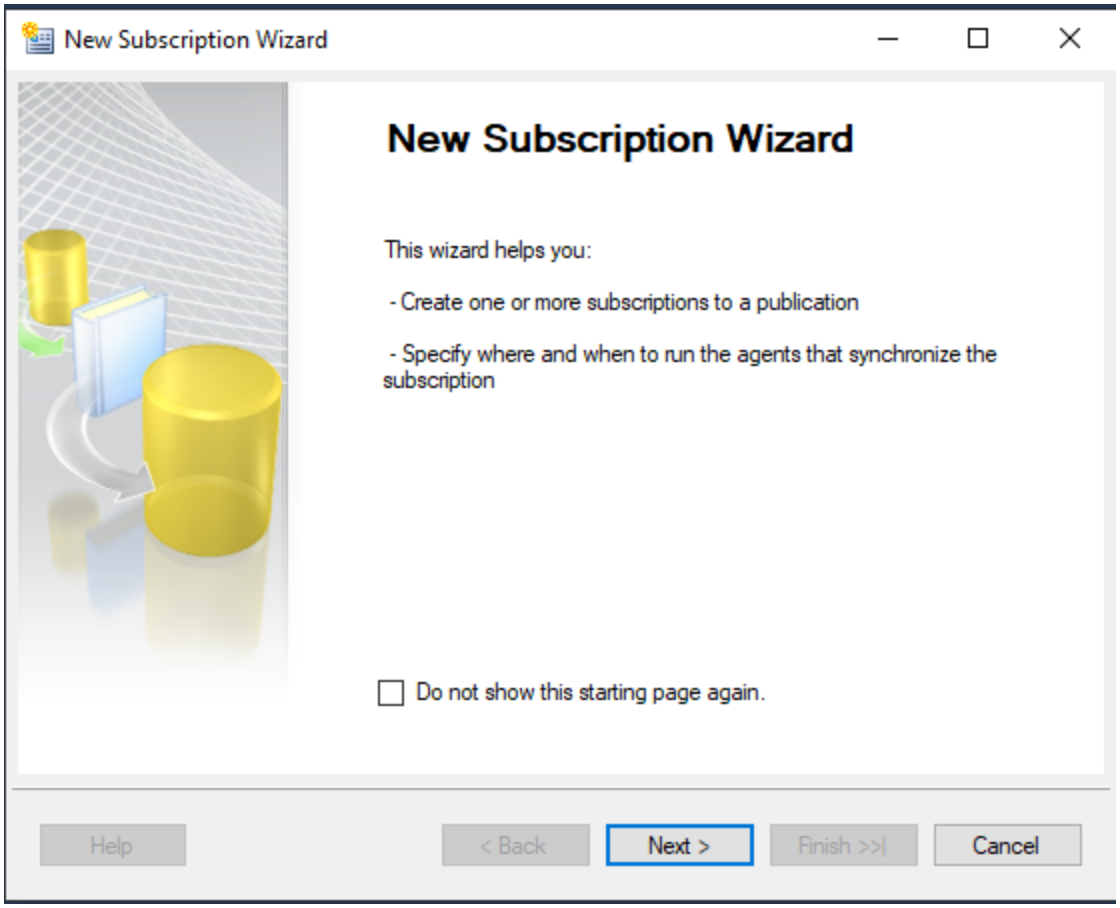
The result may look like this:



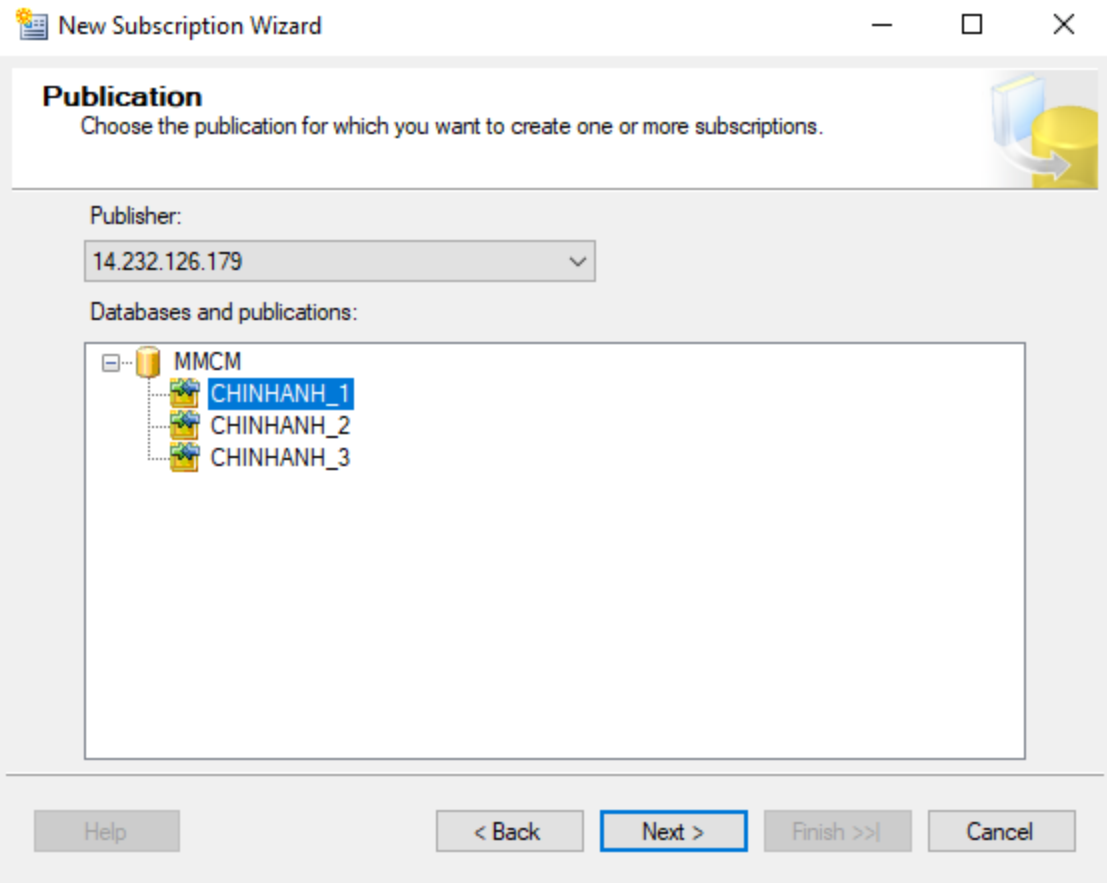
Add Subscription

Follow these steps to add subscriptions for syncing later:

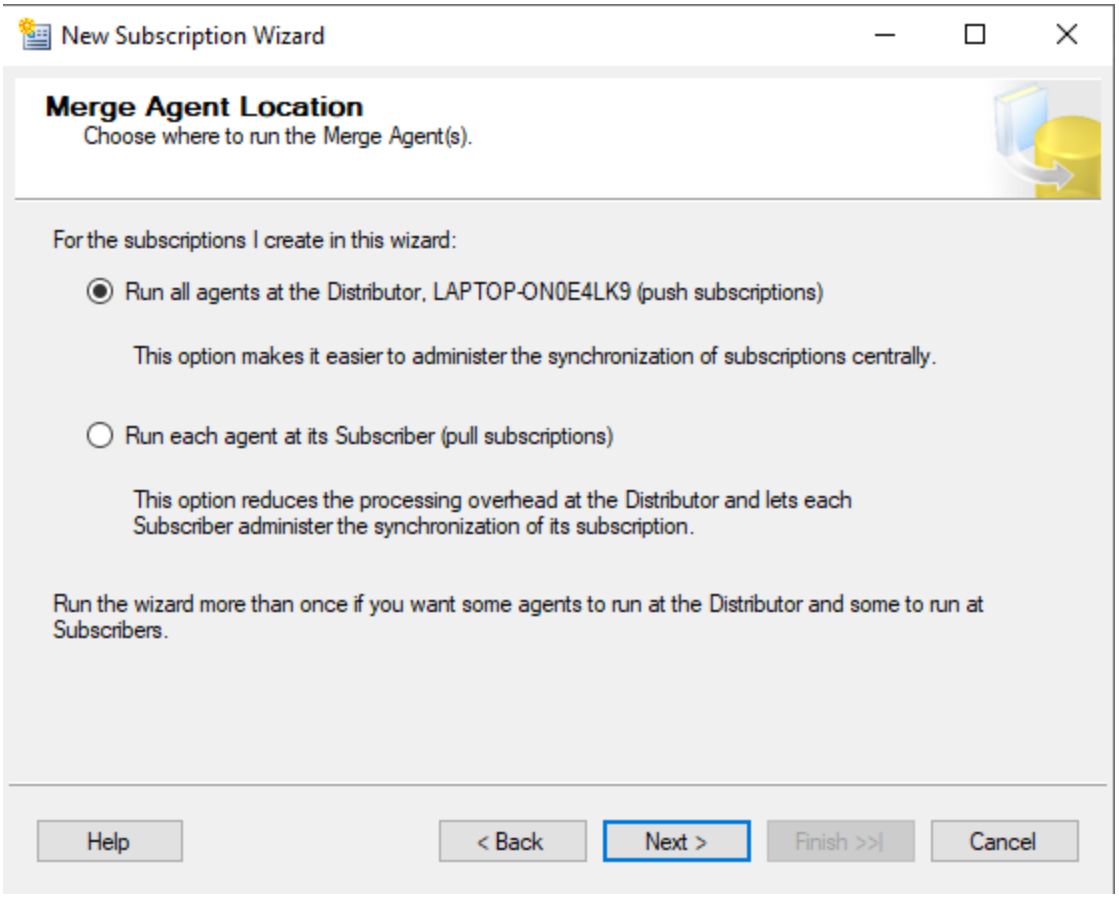

Security
Logins
##MS_PolicyEventProcessingLogin##
##MS_PolicyTsqlExecutionLogin##
distributor_admin
LAPTOP-ON0E4LK9\typpo
NT AUTHORITY\SYSTEM
NT Service\MSSQLSERVER
NT SERVICE\SQLSERVERAGENT
NT SERVICE\SQLTELEMETRY
NT SERVICE\SQLWriter
NT SERVICE\Winmgmt
sa
Server Roles
Credentials
Cryptographic Providers
Audits
Server Audit Specifications
Server Objects
Replication
Local Publications
PolyBase
Management
XEvent Profiler
New Subscriptions...
Generate Scripts...
Publisher Properties...
Distributor Properties...
Reports
Refresh



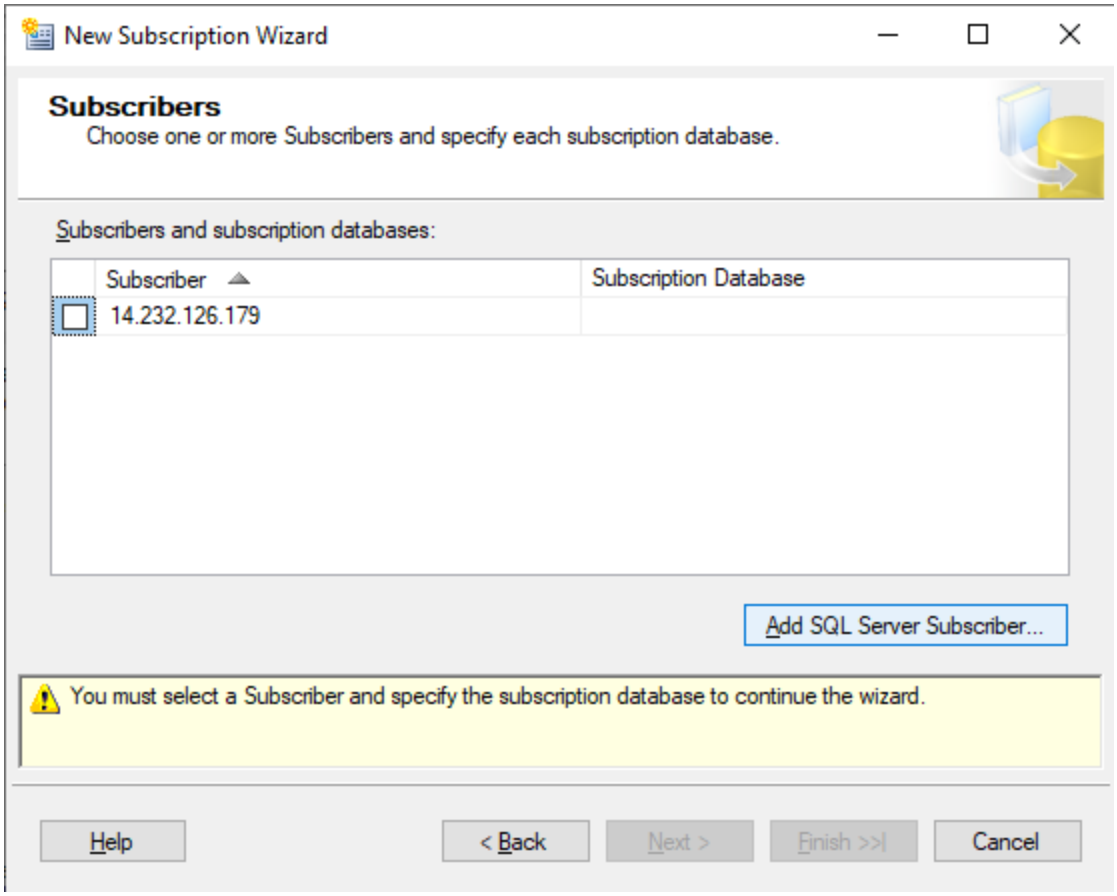
Click “Next” and choose the publication for the subscription, then click “Next”



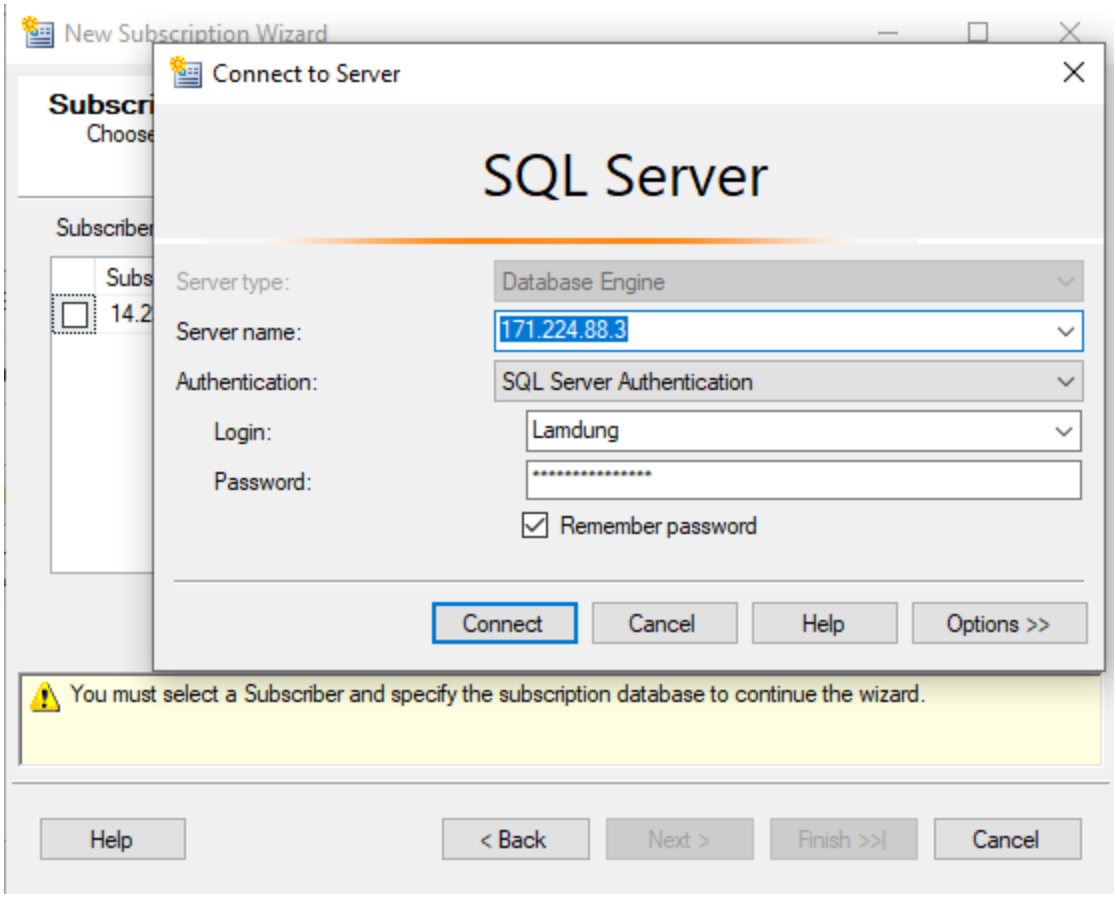
Select like below -> click “Next”



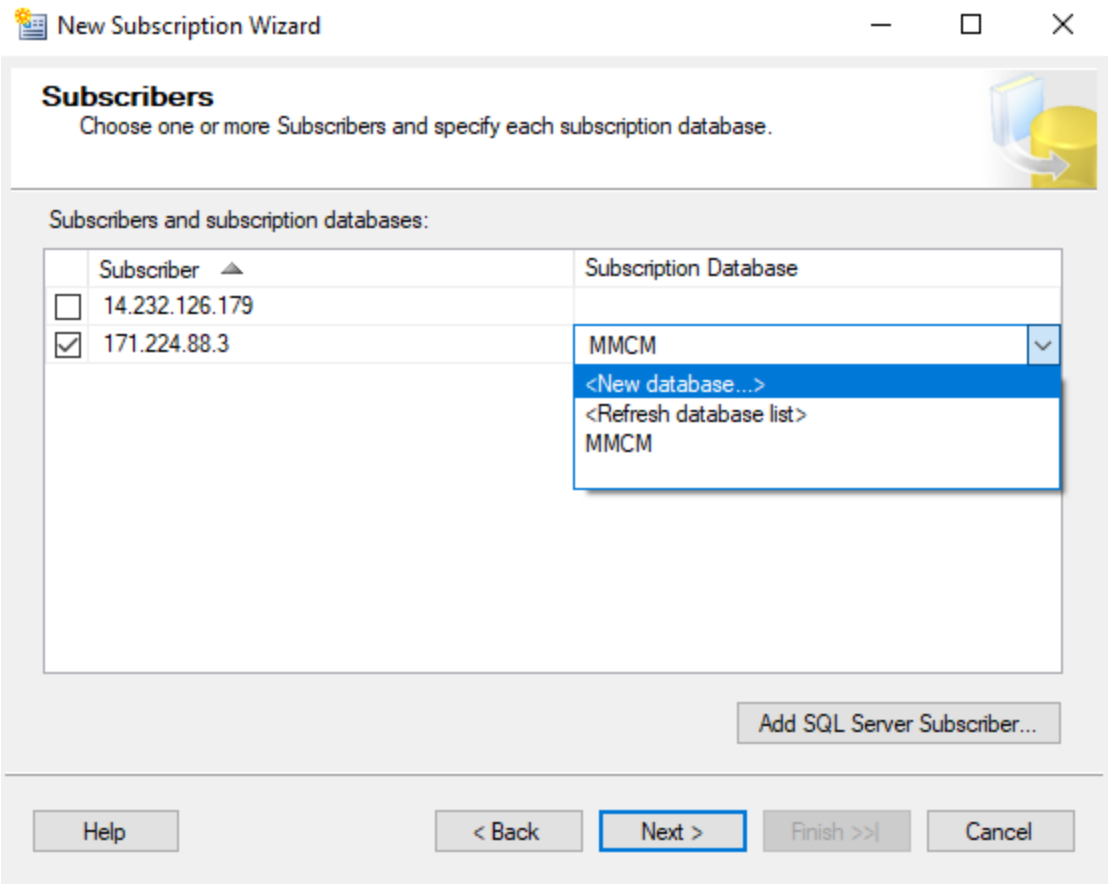
Click “Add SQL Server subscriber” -> enter server details and user details -> click “connect”



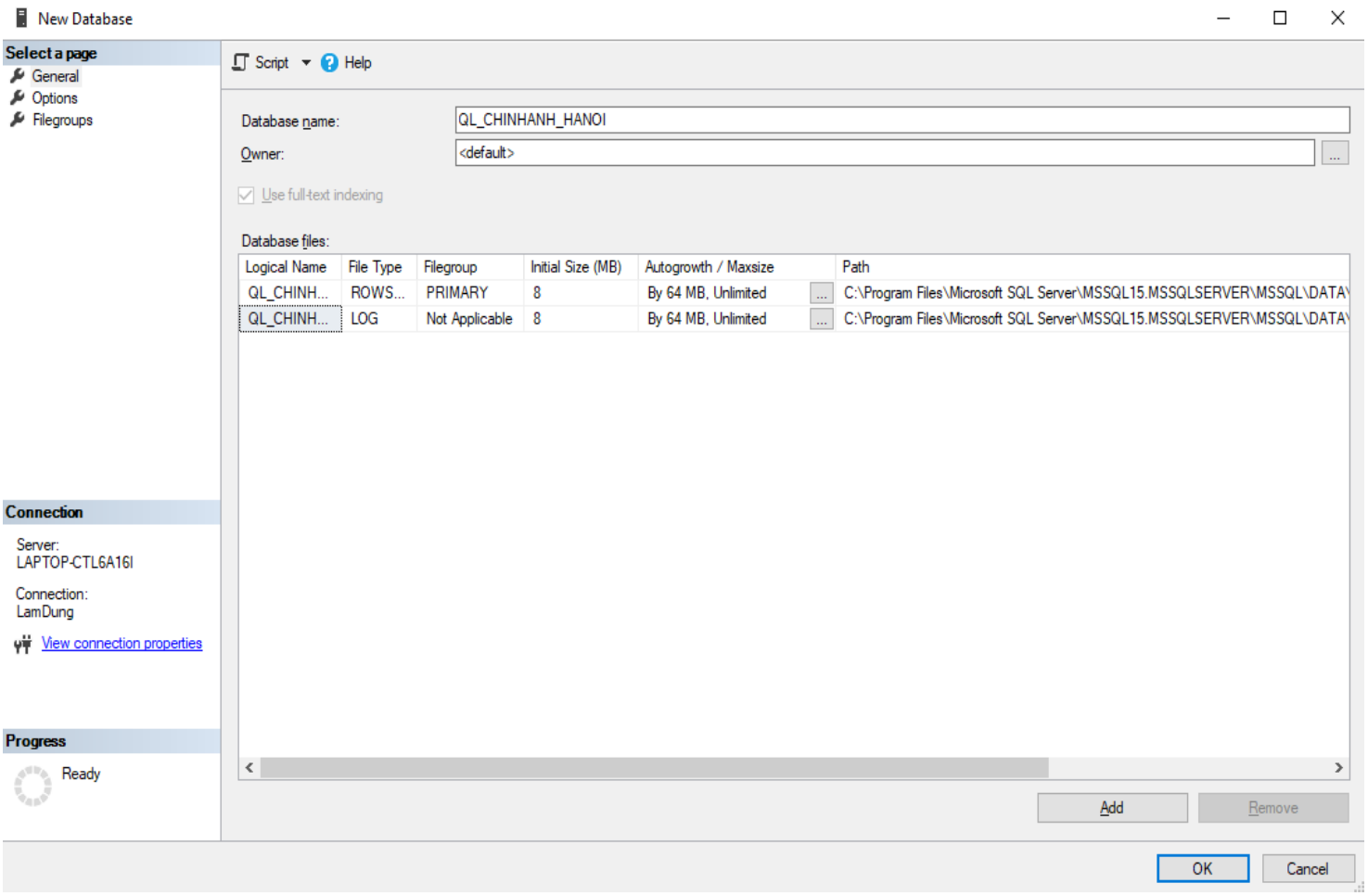
Enter the Subscriber's server's public IP Adress, username and password (using "what is my ip" search query on Google), make sure the IP is right, and the port for the Subscriber's is all opened previously

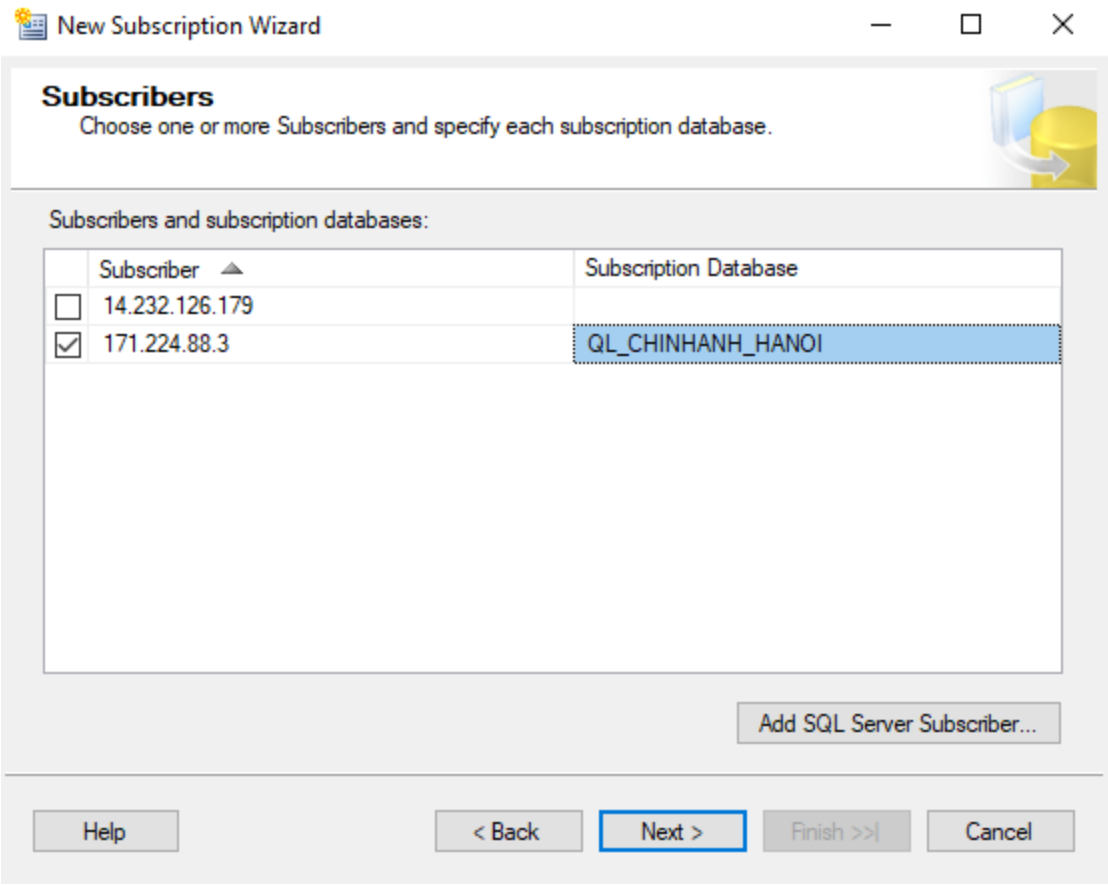


Click “Connect”, then create the database for storing the fragments

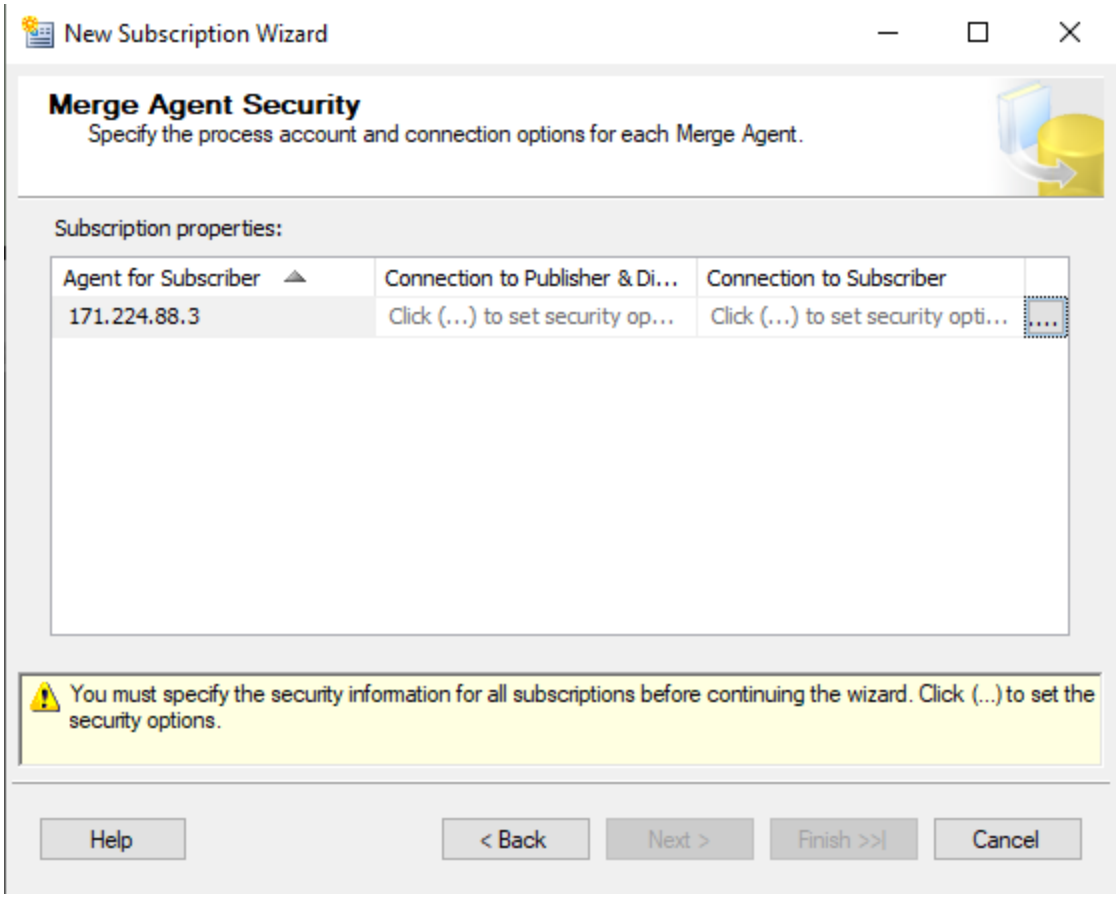


Enter the db name -> click “OK”


New Database
Select a page
General
Options
Filegroups
Script
Help
Database name:
QL_CHINHANH_HANOI
Owner:
<default>
Use full-text indexing
Database files:
Logical Name
File Type
Filegroup
Initial Size (MB)
Autogrowth / Maxsize
Path
QL_CHINH...
ROWS...
PRIMARY
8
By 64 MB, Unlimited
C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\
QL_CHINH...
LOG
Not Applicable
8
By 64 MB, Unlimited
C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\
Connection
Server:
LAPTOP-CTL6A16I
Connection:
LamDung
View connection properties
Progress
Ready
Add
Remove
OK
Cancel



Click “Next” and click “...”



Choose radiobox like below, enter the subscription user details -> Click “OK”

Merge Agent Security

Specify the domain or machine account under which the Merge Agent process will run when synchronizing this subscription.

○ Run under the following Windows account:

Process account:

Example: domain\account

Password:

Confirm Password:

◉ Run under the SQL Server Agent service account (This is not a recommended security best practice.)

Connect to the Publisher and Distributor

◉ By impersonating the process account

○ Using a SQL Server login

The connection to the server on which the agent runs must impersonate the process account. The process account must be a member of the Publication Access List.

Connect to the Subscriber

○ By impersonating the process account

◉ Using the following SQL Server login:

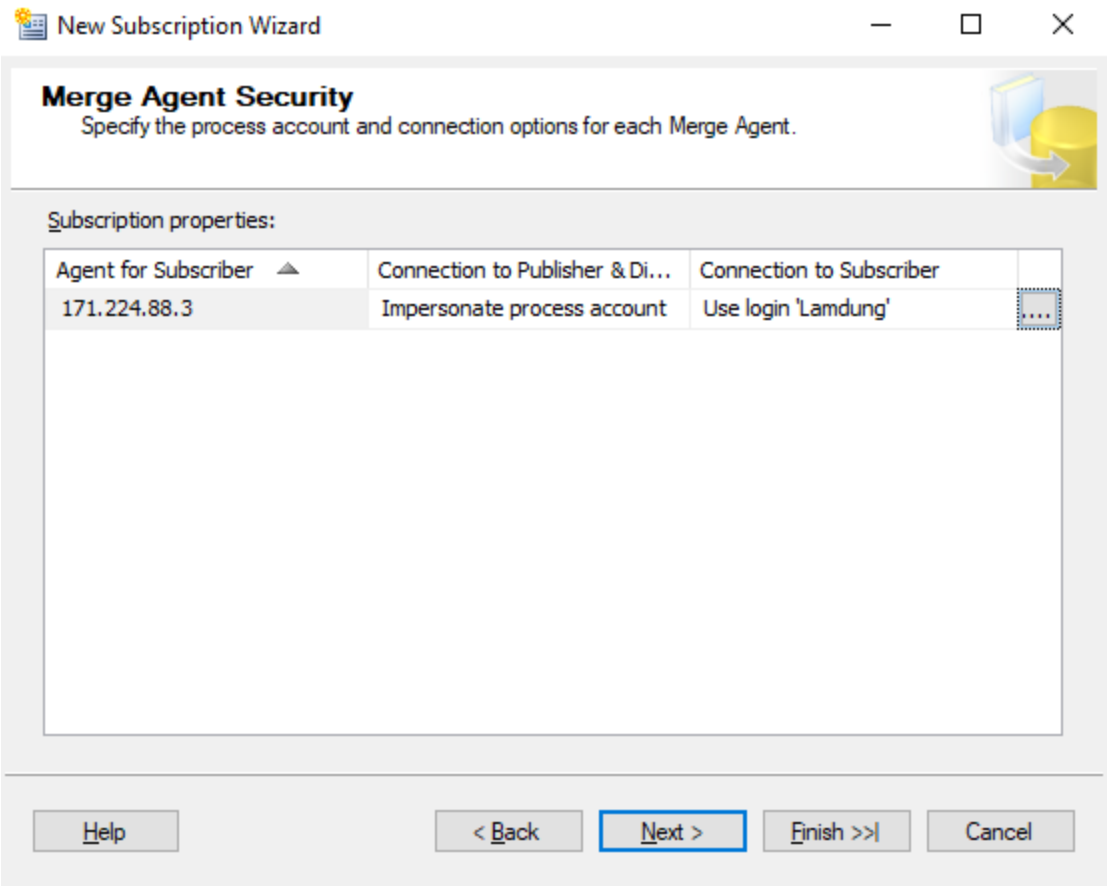
Login: Lamdung

Password: ***

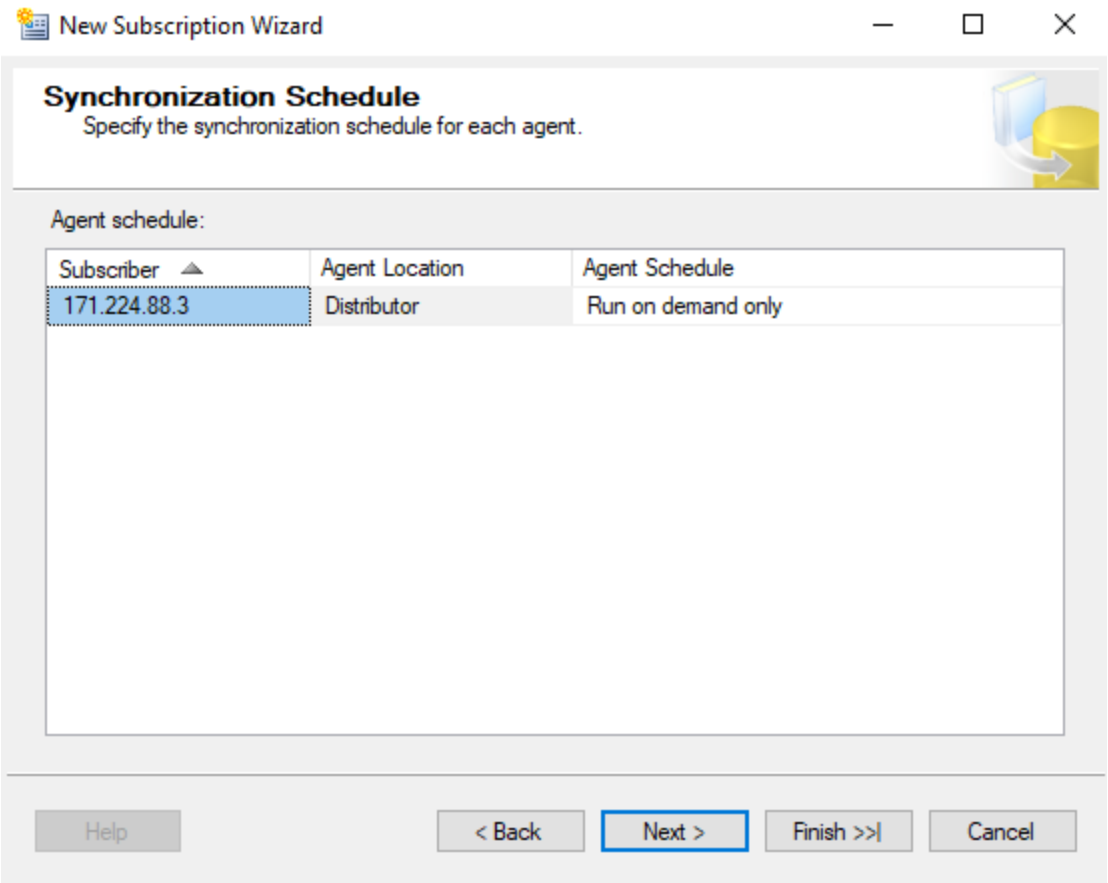
Confirm password: ***

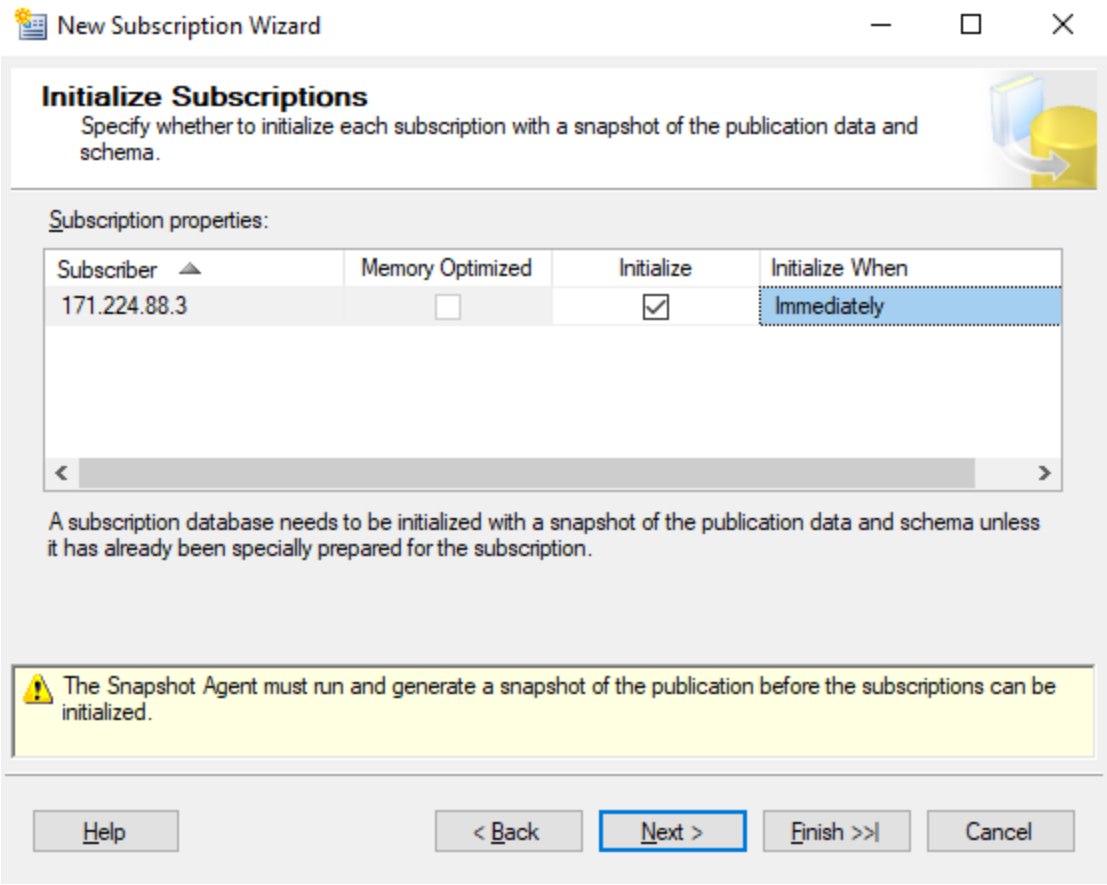
The login used to connect to the Subscriber must be a database owner of the subscription database.

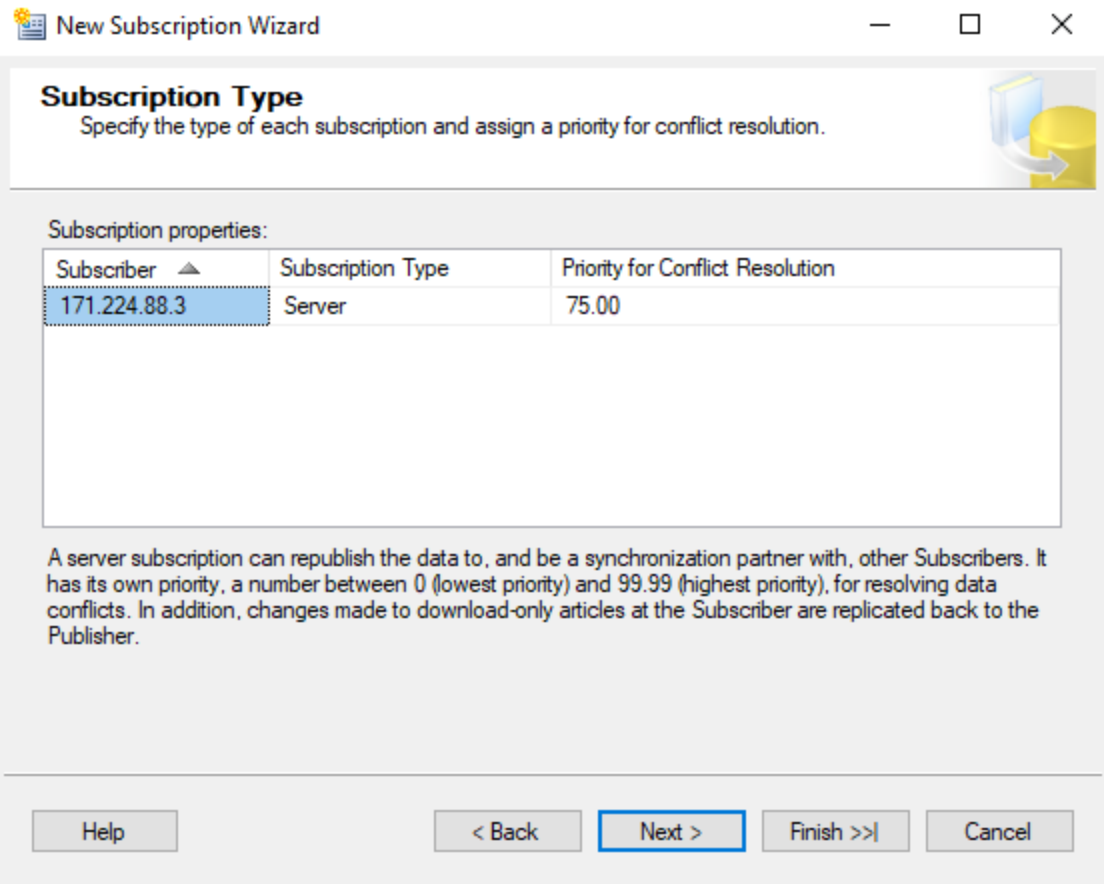
OK Cancel Help

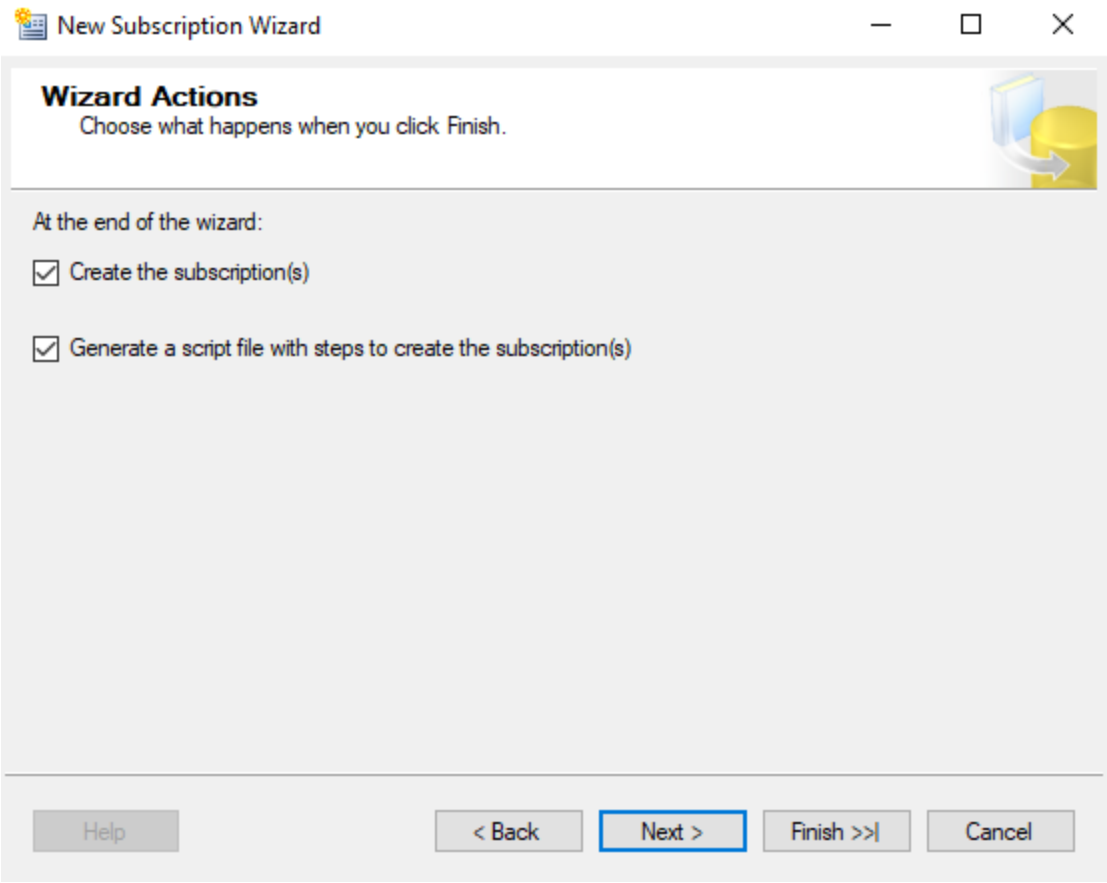


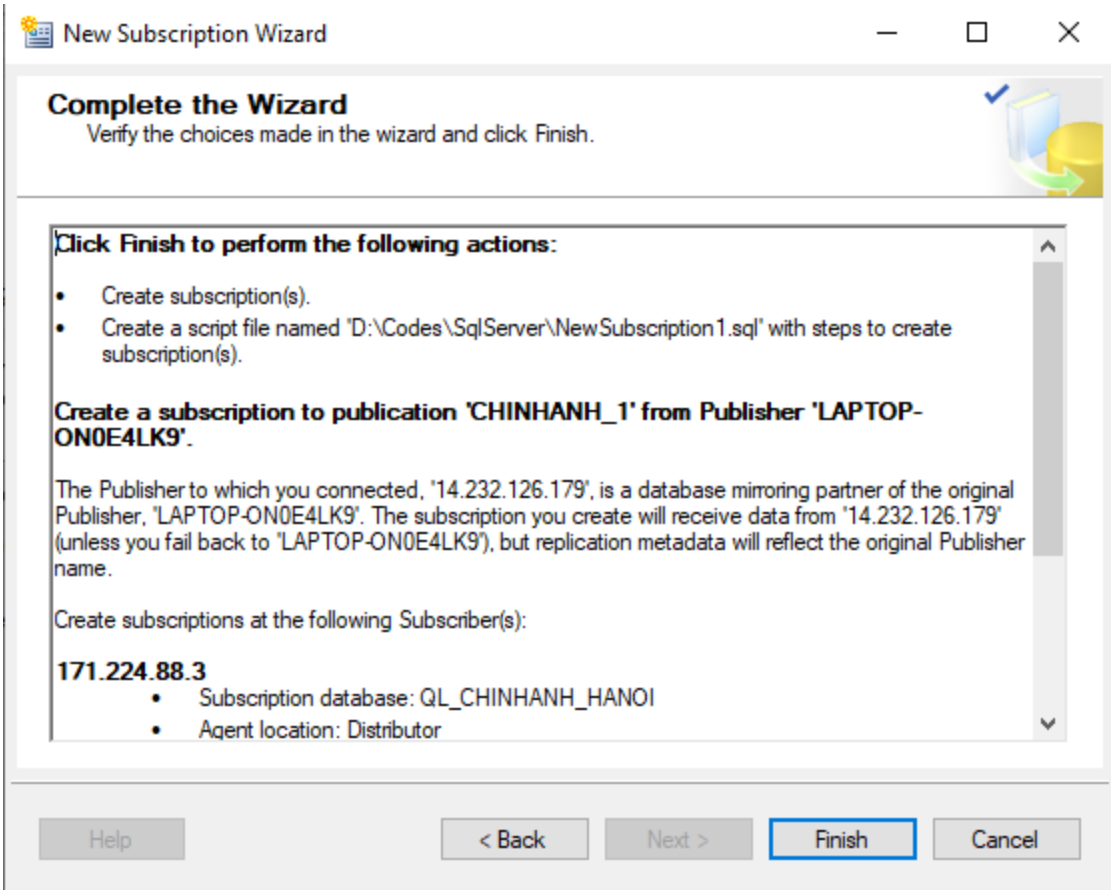
Keep following until the button “Finish”


New Subscription Wizard
Synchronization Schedule
Specify the synchronization schedule for each agent.
Agent schedule:
Subscriber
Agent Location
Agent Schedule
171.224.88.3
Distributor
Run on demand only
Help
< Back
Next >
Finish >>|
Cancel

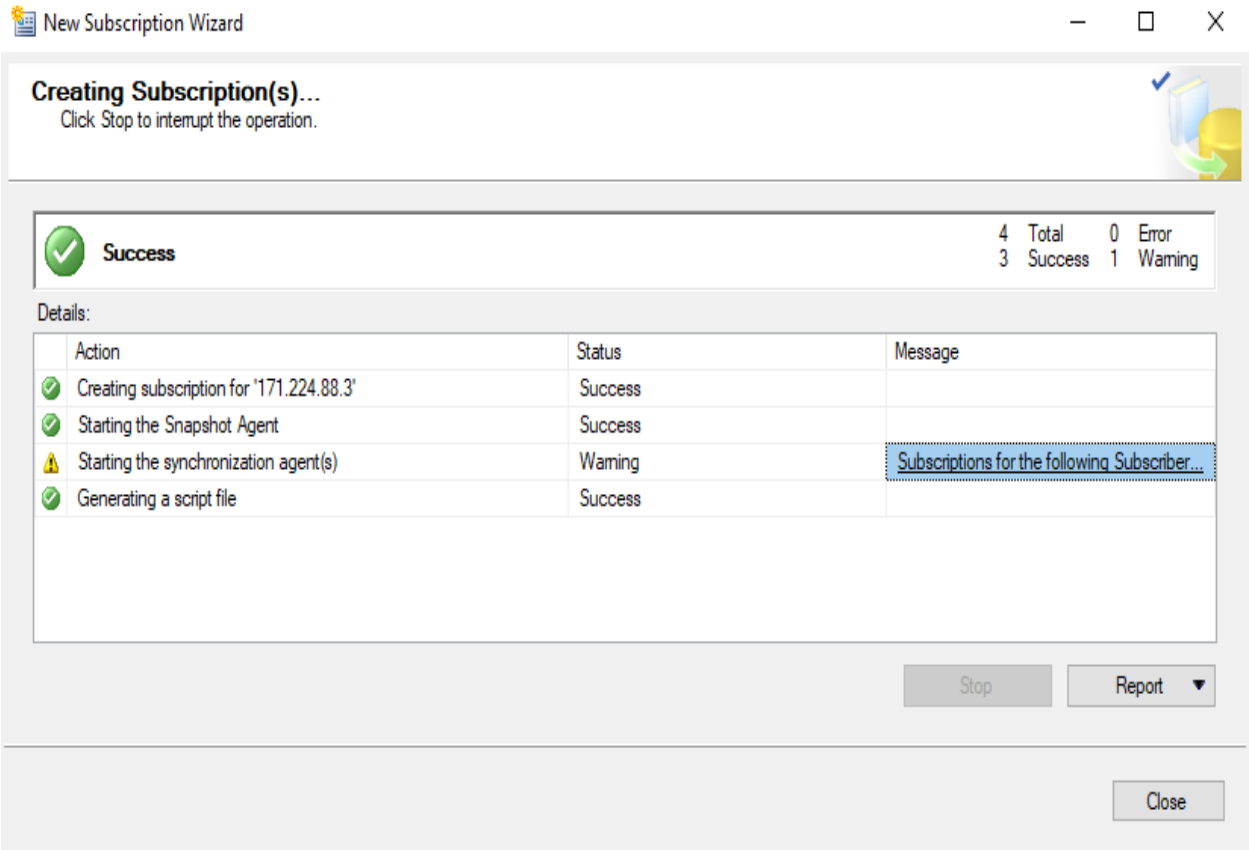

New Subscription Wizard
Initialize Subscriptions
Specify whether to initialize each subscription with a snapshot of the publication data and schema.
Subscription properties:
Subscriber
Memory Optimized
Initialize
Initialize When
171.224.88.3
Immediately
A subscription database needs to be initialized with a snapshot of the publication data and schema unless it has already been specially prepared for the subscription.
The Snapshot Agent must run and generate a snapshot of the publication before the subscriptions can be initialized.
Help
< Back
Next >
Finish >>|
Cancel


New Subscription Wizard
Subscription Type
Specify the type of each subscription and assign a priority for conflict resolution.
Subscription properties:
Subscriber
Subscription Type
Priority for Conflict Resolution
171.224.88.3
Server
75.00
A server subscription can republish the data to, and be a synchronization partner with, other Subscribers. It has its own priority, a number between 0 (lowest priority) and 99.99 (highest priority), for resolving data conflicts. In addition, changes made to download-only articles at the Subscriber are replicated back to the Publisher.
Help
< Back
Next >
Finish >>|
Cancel


New Subscription Wizard
Wizard Actions
Choose what happens when you click Finish.
At the end of the wizard:
Create the subscription(s)
Generate a script file with steps to create the subscription(s)
Help
< Back
Next >
Finish >>|
Cancel

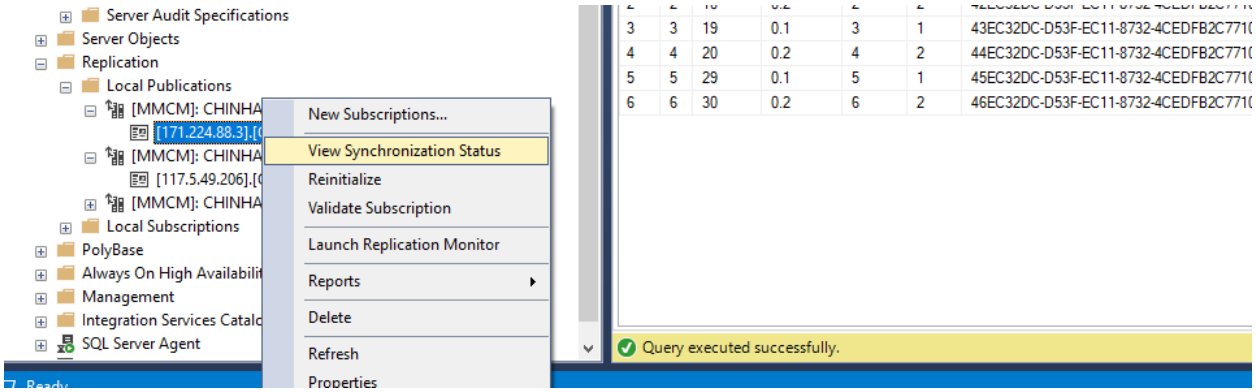


The result may look like this:

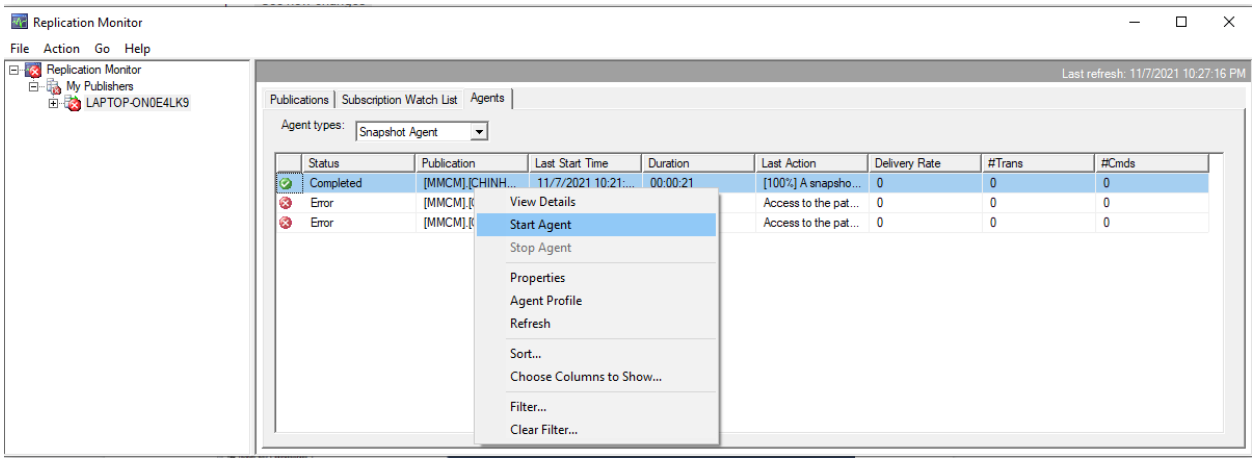


Start Synchronization

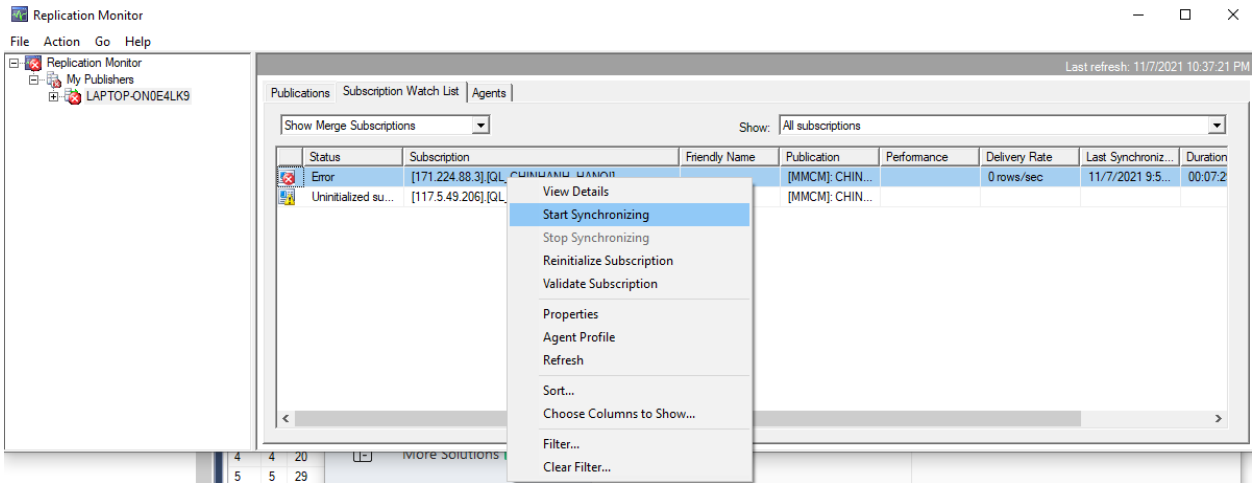
Click on the subscription and choose “View Synchronization Status”



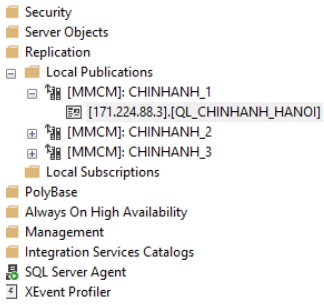
Then “Agents” -> choose Agent -> “Start Agent”

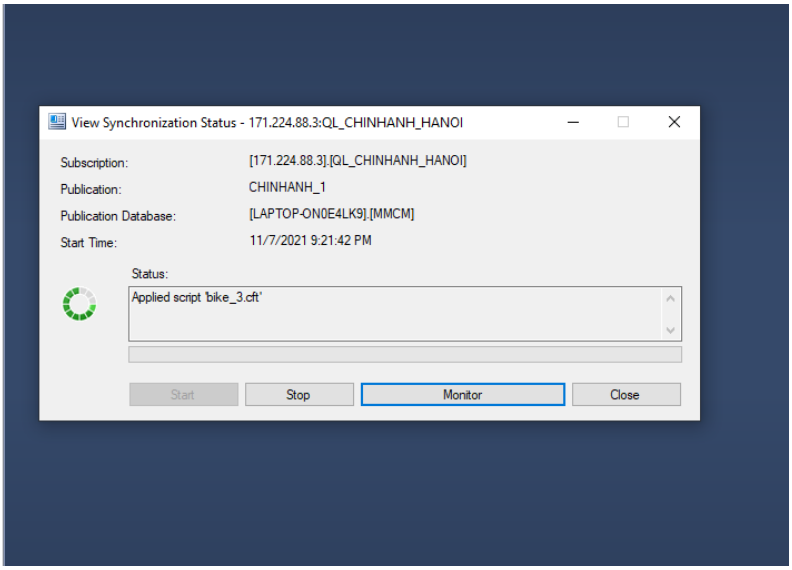


After starting Agent successfully -> “Subscription Watch List” -> choose subscription -> “Start synchronizing”



Wait for the syncing to finish





Result

The database is distributed successfully

Practice with database

Update data from Site4 and test the synchronization in Site1 (a subscription)

In Site4.MMCM:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	Danang	01234568	branch2@gmail.com	008E8205-6940-EC11-8733-4CEDFB2C7710
3	3	Hochiminh	01234569	branch3@gmail.com	018E8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710

In Site1.QL_CHINHANH_1:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710

Table account:

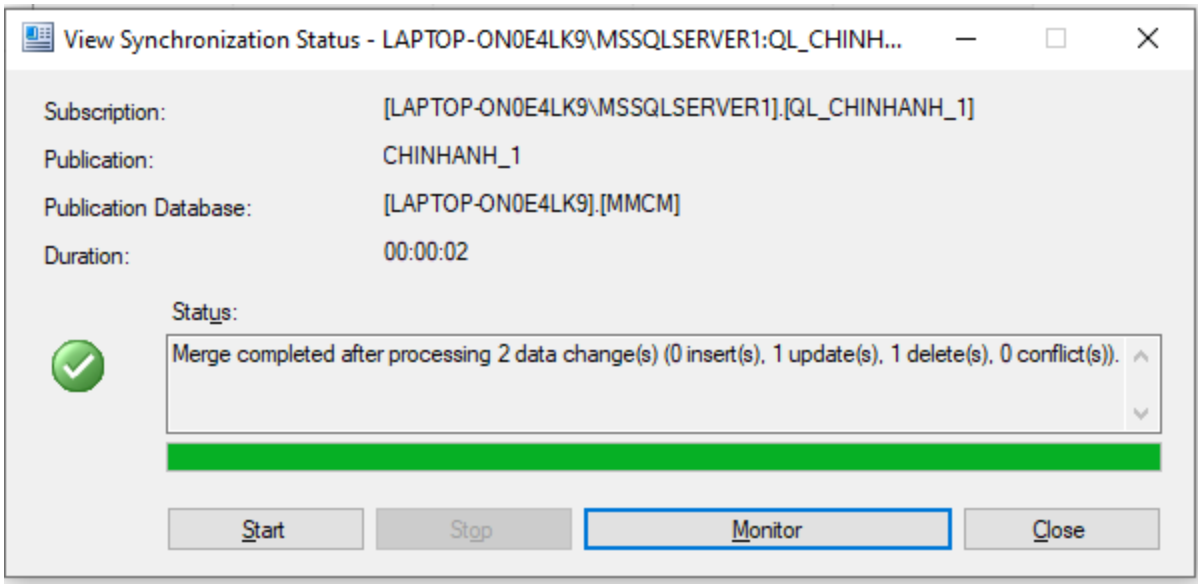
	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710

Query

Query: UPDATE table Site4.MMCM.account (change username “user1” -> “Doe”)

	id	username	password	role	rowguid
	1	doe	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
	3	user3	3	staff	bc4f9a5d-a140-...
	4	user4	4	manager	bd4f9a5d-a140-...
	5	user5	5	staff	be4f9a5d-a140-...
	6	user6	6	manager	bf4f9a5d-a140-...
*	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



In Site1.QL_CHINHANH_1.account:

	id	username	password	role	rowguid
1	1	doe	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710

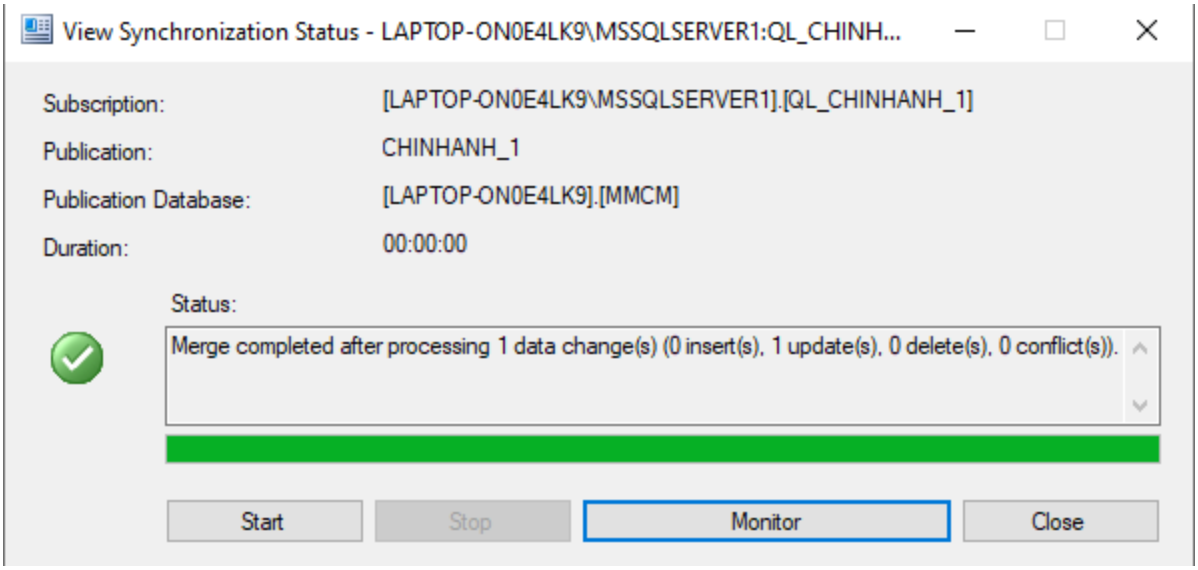
Update data from Site1 and test the synchronization in Site4

Query

Query: UPDATE table Site1.QL_CHINHANH_1.account (change username “Doe” -> “user1”)

	id	username	password	role	rowguid
	1	user1	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
*	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



In Site4.MMCM.account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710

Insert data from Site4 and test the synchronization in Site1 (a subscription)

In Site4.MMCM:

Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	Danang	01234568	branch2@gmail.com	008E8205-6940-EC11-8733-4CEDFB2C7710
3	3	Hochiminh	01234569	branch3@gmail.com	018E8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710

In Site1.QL_CHINHANH_1:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710

Query

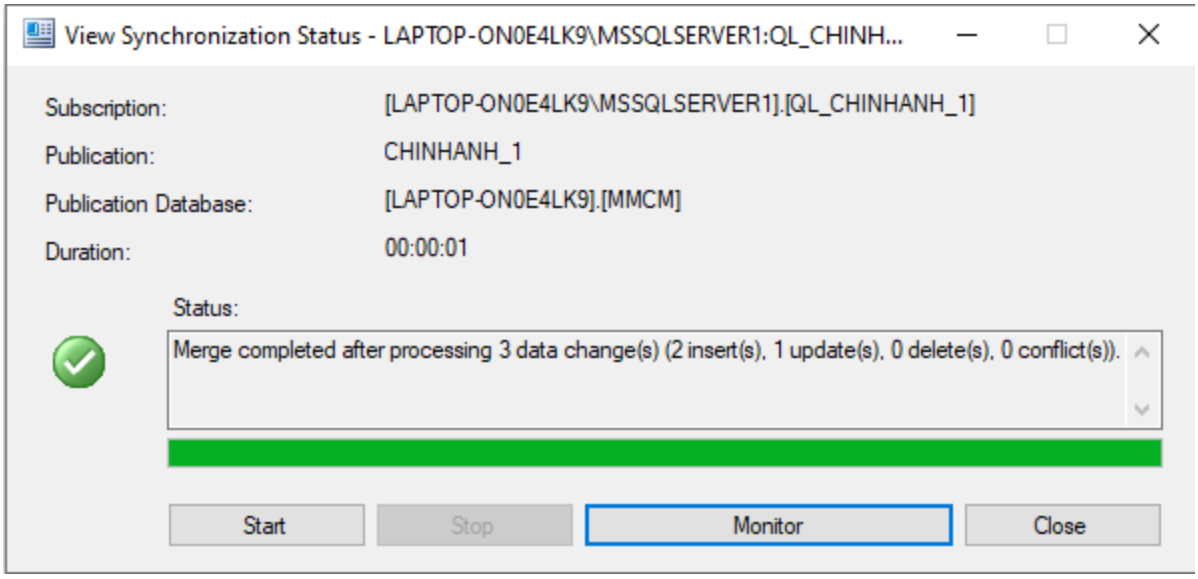
Query: INSERT table Site4.MMCM.account (add “user7”)

	id	username	password	role	rowguid
▶	1	user1	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
	3	user3	3	staff	bc4f9a5d-a140-...
	4	user4	4	manager	bd4f9a5d-a140-...
	5	user5	5	staff	be4f9a5d-a140-...
	6	user6	6	manager	bf4f9a5d-a140-...
	7	user7	7	staff	b5785a21-7141-...
*	NULL	NULL	NULL	NULL	NULL

Query: INSERT table Site4.MMCM.staff (add into “staff”)

	id	name	gender	address	email	salary	accountid	branchid	rowguid
▶	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@g...	8	1	1	198e8205-6940-...
	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@g...	8	3	2	e4b923bd-4d41...
	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@...	8	5	3	26572eed-4d41...
	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@...	8	7	1	86609157-7141-...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



In Site1.QL_CHINHANH_1.account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	7	user7	7	staff	B5785A21-7141-EC11-8734-4CEDFB2C7710

In Site1.QL_CHINHANH_1.staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@gmail.com	8	7	1	86609157-7141-EC11-8734-4CEDFB2C7710

Insert data from Site1 and test the synchronization in Site4

In Site4.MMCM:

Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	Danang	01234568	branch2@gmail.com	008E8205-6940-EC11-8733-4CEDFB2C7710
3	3	Hochiminh	01234569	branch3@gmail.com	018E8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710

In Site1.QL_CHINHANH_1:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanh Xuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710

False adding procedure:

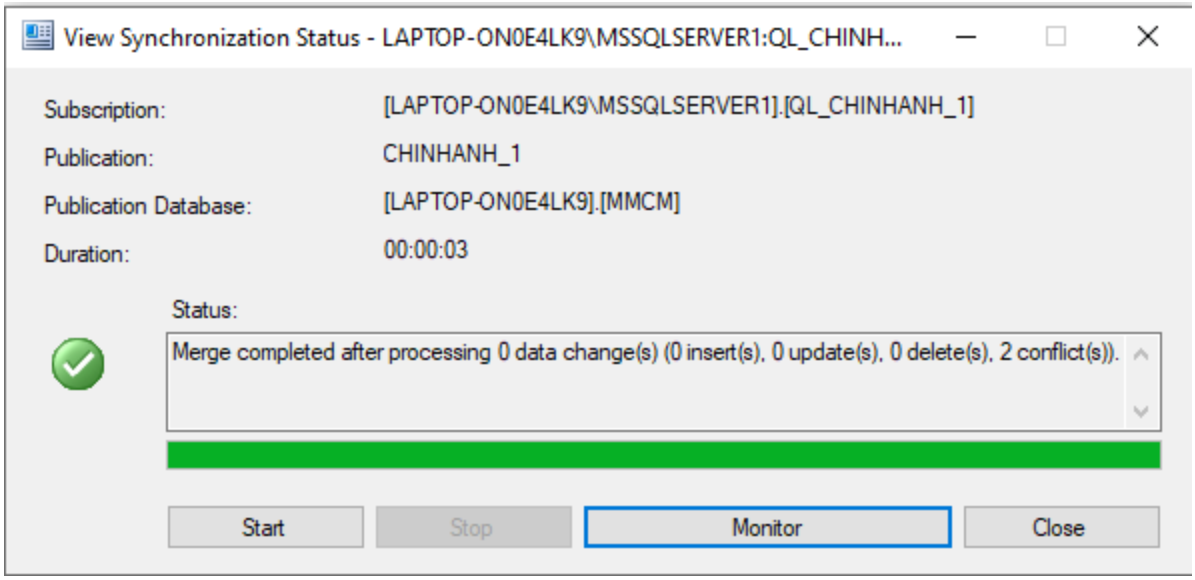
Query: INSERT table Site1.QL_CHINHANH_1.account (add “user3”)

	id	username	password	role	rowguid
▶	1	user1	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
	3	user3	3	staff	04a5a24c-7341-...
*	NULL	NULL	NULL	NULL	NULL

Query: INSERT table Site1.QL_CHINHANH_1.staff (add into “staff”)

	id	name	gender	address	email	salary	accountid	branchid	rowguid
▶	1	Nguyen Van A	Male	Thanhxuan, Ha...	nguyenvana@g...	8	1	1	198e8205-6940-...
	2	Nguyen Van B	Female	Thanhxuan, Ha...	nguyevanb@g...	8	3	1	ac84b188-7341-...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



Nothing's changed, only 2 conflicts! The reason is because of the conflicted id(s)
-> Delete the conflicted datas to resolve conflicts

Correct adding procedure:

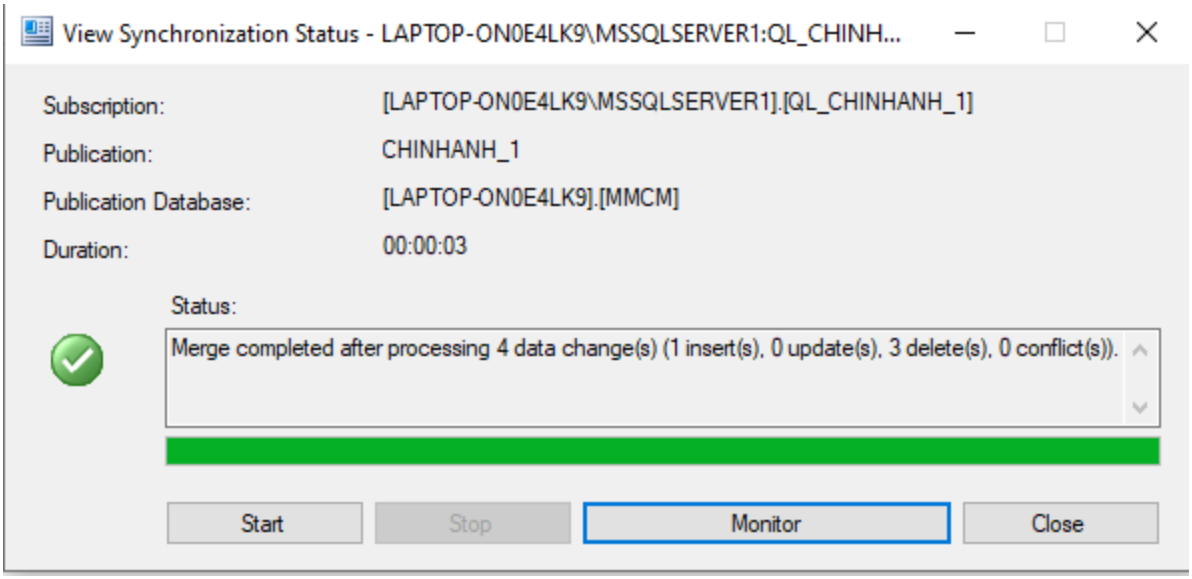
Query: INSERT table Site1.QL_CHINHANH_1.account (add "user7")

id	username	password	role	rowguid
1	user1	1	staff	fb8d8205-6940-...
2	user2	2	manager	fc8d8205-6940-...
7	user7	7	staff	04a5a24c-7341-...
NULL	NULL	NULL	NULL	NULL

Query: INSERT table Site1.QL_CHINHANH_1.staff (add into "staff")

id	name	gender	address	email	salary	accountid	branchid	rowguid
1	Nguyen Van A	Male	Thanhxuan, Ha...	nguyenvana@g...	8	1	1	198e8205-6940-...
4	Nguyen Van B	Female	Thanhxuan, Ha...	nguyenvanb@...	8	7	1	831ce91f-7541-...
NULL	NULL	NULL	NULL	NULL	NULL	NULL		NULL

Click Start merge, the result looks like this:



In Site4.MMCM.account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710
7	7	user7	7	staff	A4AC2009-7541-EC11-8734-4CEDFB2C7710

In Site4.MMCM.staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710

Delete data from Site4 and test the synchronization in Site1 (a subscription)

In Site4.MMCM:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	Danang	01234568	branch2@gmail.com	008E8205-6940-EC11-8733-4CEDFB2C7710
3	3	Hochiminh	01234569	branch3@gmail.com	018E8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710
4	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@gmail.com	8	7	1	86609157-7141-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710
7	7	user7	7	staff	B5785A21-7141-EC11-8734-4CEDFB2C7710

In Site1.QL_CHINHANH_1:

Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@gmail.com	8	7	1	86609157-7141-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	7	user7	7	staff	B5785A21-7141-EC11-8734-4CEDFB2C7710

Query

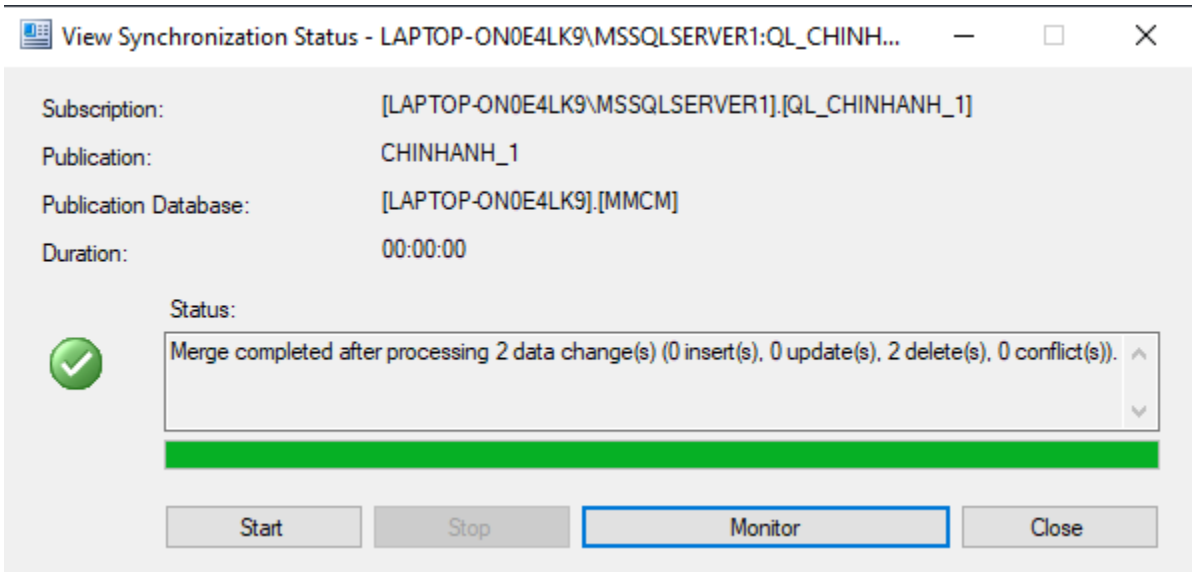
Query: DELETE table Site4.MMCM.staff (delete staff with accountid = 7)

	id	name	gender	address	email	salary	accountid	branchid	rowguid
▶	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@g...	8	1	1	198e8205-6940-...
	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@g...	8	3	2	e4b923bd-4d41...
	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@...	8	5	3	26572eed-4d41...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Query: DELETE table Site4.MMCM.account (delete “user7”)

	id	username	password	role	rowguid
▶	1	user1	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
	3	user3	3	staff	bc4f9a5d-a140-...
	4	user4	4	manager	bd4f9a5d-a140-...
	5	user5	5	staff	be4f9a5d-a140-...
	6	user6	6	manager	bf4f9a5d-a140-...
*	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



In Site1.QL_CHINHANH_1.account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710

In Site1.QL_CHINHANH_1.staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710

Delete data from Site1 and test the synchronization in Site4

In Site4.MMCM:
Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	Danang	01234568	branch2@gmail.com	008E8205-6940-EC11-8733-4CEDFB2C7710
3	3	Hochiminh	01234569	branch3@gmail.com	018E8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710
4	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@gmail.com	8	7	1	86609157-7141-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710
7	7	user7	7	staff	B5785A21-7141-EC11-8734-4CEDFB2C7710

In Site1.QL_CHINHANH_1:

Table branch:

	id	address	tel	email	rowguid
1	1	Hanoi	01234567	branch1@gmail.com	FF8D8205-6940-EC11-8733-4CEDFB2C7710

Table staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	4	Nguyen Van B	Male	Thanhxuan, Hanoi	nguyenvanb@gmail.com	8	7	1	86609157-7141-EC11-8734-4CEDFB2C7710

Table account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	7	user7	7	staff	B5785A21-7141-EC11-8734-4CEDFB2C7710

Query

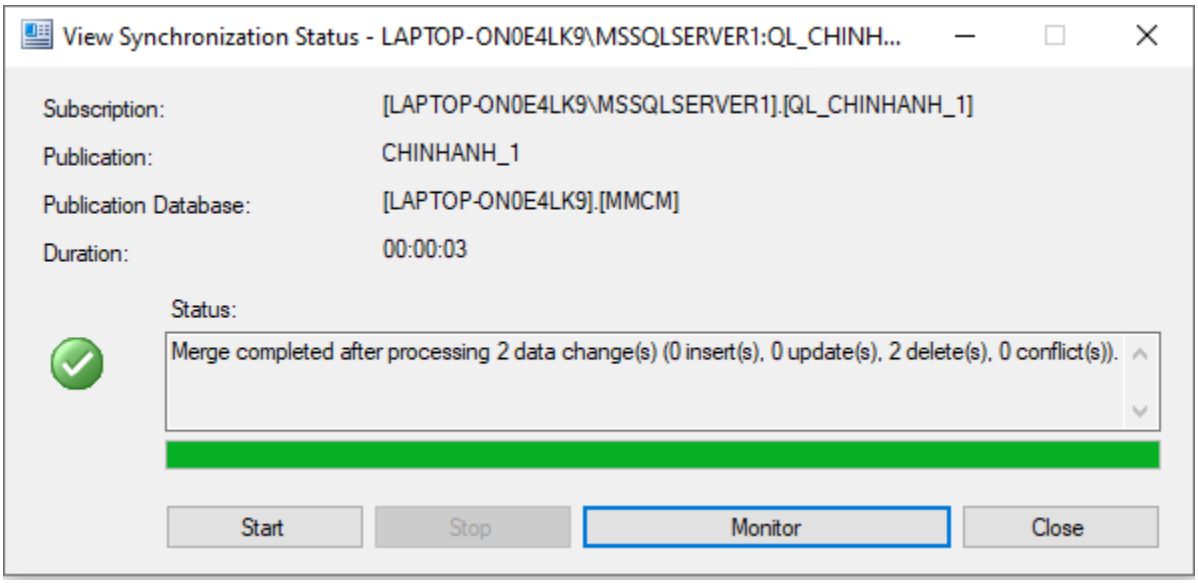
Query: DELETE table Site1.QL_CHINHANH_1.staff (delete staff with accountid = 7)

	id	name	gender	address	email	salary	accountid	branchid	rowguid
▶	1	Nguyen Van A	Male	Thanhxuan, Ha...	nguyenvana@g...	8	1	1	198e8205-6940-...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Query: DELETE table Site1.QL_CHINHANH_1.account (delete “user7”)

	id	username	password	role	rowguid
▶	1	user1	1	staff	fb8d8205-6940-...
	2	user2	2	manager	fc8d8205-6940-...
*	NULL	NULL	NULL	NULL	NULL

Click Start merge, the result looks like this:



In Site4.MMCM.account:

	id	username	password	role	rowguid
1	1	user1	1	staff	FB8D8205-6940-EC11-8733-4CEDFB2C7710
2	2	user2	2	manager	FC8D8205-6940-EC11-8733-4CEDFB2C7710
3	3	user3	3	staff	BC4F9A5D-A140-EC11-8733-4CEDFB2C7710
4	4	user4	4	manager	BD4F9A5D-A140-EC11-8733-4CEDFB2C7710
5	5	user5	5	staff	BE4F9A5D-A140-EC11-8733-4CEDFB2C7710
6	6	user6	6	manager	BF4F9A5D-A140-EC11-8733-4CEDFB2C7710

In Site4.MMCM.staff:

	id	name	gender	address	email	salary	accountid	branchid	rowguid
1	1	Nguyen Van A	Male	Thanhxuan, Hanoi	nguyenvana@gmail.com	8	1	1	198E8205-6940-EC11-8733-4CEDFB2C7710
2	2	Nguyen Thi A	Female	Haichau, Danang	nguyenthia@gmail.com	8	3	2	E4B923BD-4D41-EC11-8734-4CEDFB2C7710
3	3	Nguyen Tien A	Female	Govap, Hochiminh	nguyentiena@gmail.com	8	5	3	26572EED-4D41-EC11-8734-4CEDFB2C7710

Commit protocols/Concurrency control & Distributed Failure/Recovery

Commit protocols & Concurrency control

Commit protocols & Distributed 2-phase commit protocol

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.

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

According to this article (Example: How SQL Server Explicitly Initiates a Transaction):

"When the transaction completes, the stored procedure that initiated the transaction invokes the Transact-SQL COMMIT TRANSACTION statement. SQL Server then invokes the DTC Commit method. The DTC uses the ***two-phase commit protocol*** *to coordinate commitment of the transaction. (Alternatively, the stored procedure could call the Transact-SQL ROLLBACK TRANSACTION statement. SQL Server then calls the DTC Abort method to undo the effects of the transaction)."*

Thus we will explain the distributed two-phase commit.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

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, 2PL & SS2PL

SQL Server uses strong strict two-phase locking (SS2PL), but first we need to understand 2PL.

1. 2-Phase Locking (2PL)

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

1. **Growing Phase:** New locks on data items may be acquired but none can be released.

2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	.....	.....
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	.....	.....

Transaction T1:

- The growing Phase is from steps 1-3.
- The shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T2:

- The growing Phase is from steps 2-6.
- The shrinking Phase is from steps 8-9.
- Lock Point at 6

What is 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.

2. Strict Strong 2-Phase Locking (SS2PL)

SS2PL 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.

Demo

1. **Demonstrate that 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.**

<u>Demonstrate a delayed example:</u>

We'll run our demo using the Bike table:

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	45	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	50	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Then suppose we have 2 transactions - T1 and T2
T1 (we'll update the row with id=1 to a new price=30):

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

T2:

```
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

Run T1 without committing it (select all operations except for "commit" line)

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set
wher

comm
```

Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Insert Snippet...	Ctrl+K, Ctrl+X
Surround With...	Ctrl+K, Ctrl+S
Connection	▸
Open Server in Object Explorer	Alt+F8
Execute	F5
Display Estimated Execution Plan	Ctrl+L
IntelliSense Enabled	Ctrl+B, Ctrl+I
Trace Query in SQL Server Profiler	Ctrl+Alt+P
Analyze Query in Database Engine Tuning Advisor	

Result for T1:

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

100 %

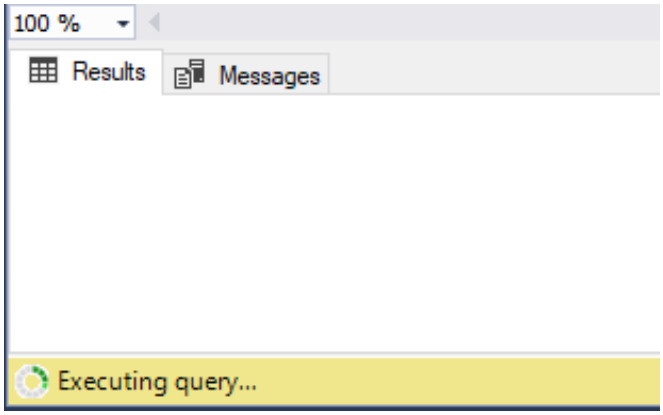
Messages

```
(1 row affected)

Completion time: 2021-11-27T10:57:52.4698028+07:00
```

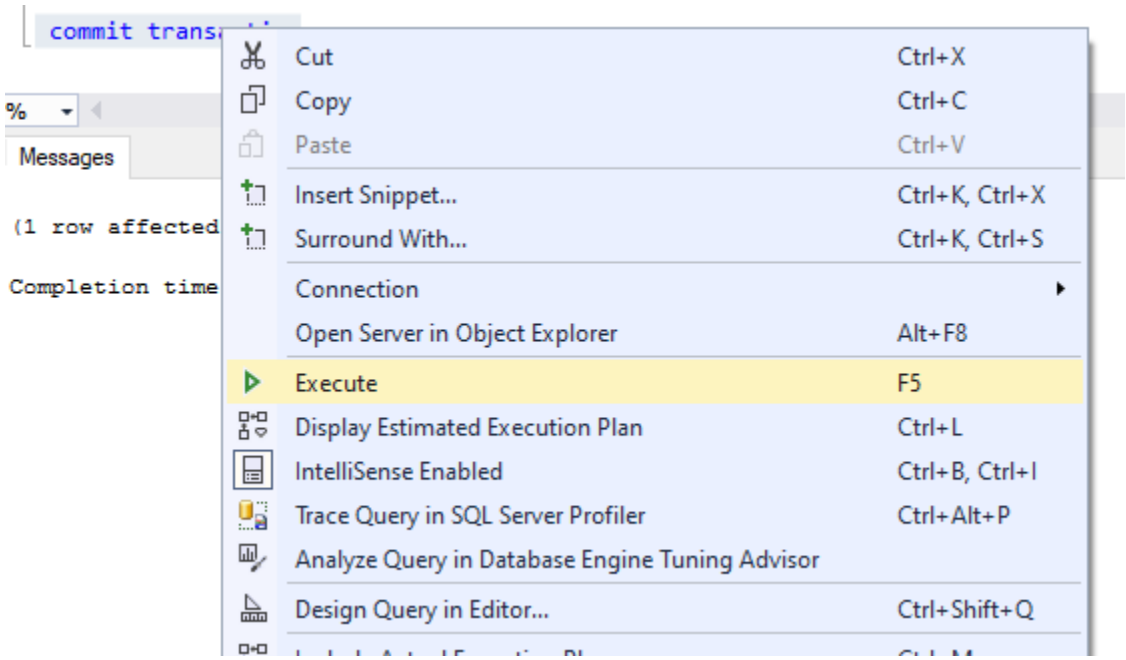
Run T2 (with/without the “commit” line - doesn’t matter):

```
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

We'll see that it's executing without any result (because it's waiting for T1 to commit)



Select the "commit transaction" line in T1 then "execute"



We'll see this message:

```
100 %
Messages
Commands completed successfully.

Completion time: 2021-11-27T11:01:48.4502644+07:00

100 %
Query executed successfully.
```

Also for T2:

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	30	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	45	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	50	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Query executed successfully.

Demonstrate an aborted example:

We still apply the above example but for this time, instead of commiting T1, we'll abort it and see how T2 runs (given T2 is waiting for T1 to commit)
Close T1 and click "No" in the dialog:

update_bike_1.sql -...LK9.master (sa (56)) | select_bike_1.sql -...4LK9.master (sa (53)) | open_tran.sql - LAP...LK9.master

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

Microsoft SQL Server Management Studio

There are uncommitted transactions. Do you wish to commit these transactions?

Copy message | Yes | No | Cancel

```
100 %
(1 row affected)
```

T2 immediately finishes:

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	45	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	50	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

We can see that the change from T1 doesn't affect the current data (price=25 instead of 30), because it hasn't yet been committed.

Conclusion: 2 transactions entered in a non-serialized order can be delayed, aborted, which is equivalent to serialized ordering.

2. Demonstrate that two transactions operating on the same tables, but different rows, can execute concurrently.

First we have the table `bike`:

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	45	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	50	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

T1: (update where id=2 set price=30)

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 2

commit transaction
```

T2: (update where id=3 set price=40)

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 40
where id = 3

commit transaction
```

Run T1 without committing

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 2

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T11:19:34.2660793+07:00
```

Run T2 without committing

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 40
where id = 3

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T11:22:50.1749733+07:00
```

Create a T3 for selecting the ‘bike’ table, and execute it

```
select_bike_1.sql - L...sa (53)) Executing...

set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction

100 %
Results   Messages
Executing query...
```

Now commit T1:

```
commit transaction

100 %
Messages
Commands completed successfully.

Completion time: 2021-11-27T11:24:47.6438476+07:00
```

T3 is still executing, so we commit T2:

```
commit transaction

100 %
Messages
Commands completed successfully.

Completion time: 2021-11-27T11:25:20.4740304+07:00
```

T3 finishes immediately

select_bike_1.sql -...4LK9.master (sa (53))

```sql
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Excit...	2021	Hon...	40	Exciter1, brand Honda, model 2...	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Query executed successfully.

Conclusion: two transactions operating on the same tables, but different rows, can execute concurrently.

3. Demonstrate that you handle read/write conflicts as well as write/write conflicts.

Read/write conflict

We'll run the example with table Bike below:

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

T1:

```sql
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

T2 (where id=1 set price =30):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

Run T1 without committing it, the result and the message look like this:

```sql
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

```sql
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Results | Messages

```
(3 rows affected)

Completion time: 2021-11-27T13:18:09.8037151+07:00
```

Execute T2 without committing it

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

100 %

Results | Messages

Executing query...

We can see that T2 waits for T1 to commit before it finishes executing

Now let's commit T1

```
SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2021-11-27T13:21:12.3243758+07:00
```

Immediately, T2 shows a message:

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T13:21:12.3513464+07:00
```

Commit T2

```
commit transaction
```

.00 %

Messages

```
Commands completed successfully.

Completion time: 2021-11-27T13:21:54.1785642+07:00
```

Now select the Bike table:

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	30	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Conclusion: Read/write conflict is handled successfully

Write/write conflict

We have the Bike table as below:

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	30	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

T1 (where id=1 set price=25):

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 25
where id = 1

commit transaction
```

T2 (where id=1 set price=30):

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 35
where id = 1

commit transaction
```

Let's run T1 without committing it:

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 25
where id = 1

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T13:28:42.4092823+07:00
```

Run T2 without committing it, we can see that T2 is now waiting for T1 to finish:

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 35
where id = 1

commit transaction
```

100 %

Results | Messages

Executing query...

Commit T1:

```sql
commit transaction
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2021-11-27T13:30:39.3327834+07:00
```

T2 immediately throws a message:

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 35
where id = 1

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T13:30:39.3656720+07:00
```

Rollback T2 and set the isolation level to read uncommitted to see the change made by T1 (price =25)

```sql
set transaction isolation level read uncommitted
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Rerun T1 and T2 like above and then commit T1; T2 now is uncommitted but try reading uncommitted data this time instead of rolling it back, the table Bike has a new result made by T2:

```sql
set transaction isolation level read uncommitted
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	35	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Now, let's commit T2 and we receive the message:

```sql
commit transaction
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2021-11-27T13:39:16.5294267+07:00
```

Run the select query with isolation level 'serializable', the changes are made into table Bike

```sql
set transaction isolation level serializable
begin transaction

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	35	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Conclusion: The Write/write conflict is handled successfully

4. **Prove that your scheme will not block indefinitely. I.e., if you have a deadlock, you must detect and correct for it. You may find that there are things that you don't handle well. As long as you demonstrate the above, it is okay if there are some things you can't handle - as long as you are aware of (and document) the conditions under which your concurrency control fails.**

Here we have our Bike table:

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	35	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

Our Part table:

	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	15	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

T1(update `bike`(id=1) set price=30 + select table ‘part’):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]

commit transaction
```

T2(update part(id=1) set price=15 + select table ‘bike’):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 15
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

Run T1(operation1):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T14:21:46.0752968+07:00
```

Run T2(operation1):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 15
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction
```

100 %

Messages

```
(1 row affected)

Completion time: 2021-11-27T14:20:35.6645670+07:00
```

Run T1(operation2):

```
deadlock_T1.sql - LA...a (63)) Executing...

set transaction isolation level serializable
begin transaction

update mmcm.dbo.bike
set price = 30
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]

commit transaction

100 %
Results   Messages

Executing query...                                  LAPTOP-ON
```

Run T2(operation 2):

```
deadlock_T2.sql -...4LK9.mmcm (sa (67))

set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 15
where id = 1

SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[bike]

commit transaction

100 %
Results   Messages
Msg 1205, Level 13, State 51, Line 8
Transaction (Process ID 67) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Completion time: 2021-11-27T14:23:02.9345780+07:00

100 %
Query completed with errors.   LAPTOP-ON0E4LK9 (15.0 RTM) | sa (67) | mmcm | 00:00:03 | 0 row
```

We can see that T1(operation2) waits for T2(operation1) to finish updating the table ‘part’.

But then T2(operation2) executes and waits for T1 to finish updating the table ‘bike’.

Both T1 and T2 then wait for each other, which gives us a deadlock.

The SQLServer then kills the transaction T2 like above to cope with deadlock.

Conclusion: SQLServer has a mechanism to prevent deadlock by terminating the transactions which caused deadlock so the others will be able to continue. We can save the terminated transactions then run it again later or report to the owners of the transactions so that they can decide what to do themself.

Distributed Failure/Recovery

Distributed Failure/Recovery

Why recovery is needed in DBMS:

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.

Demo

1. Correct recovery after a commit, to all sites having updates.

This time we'll test the Part table in a slave site

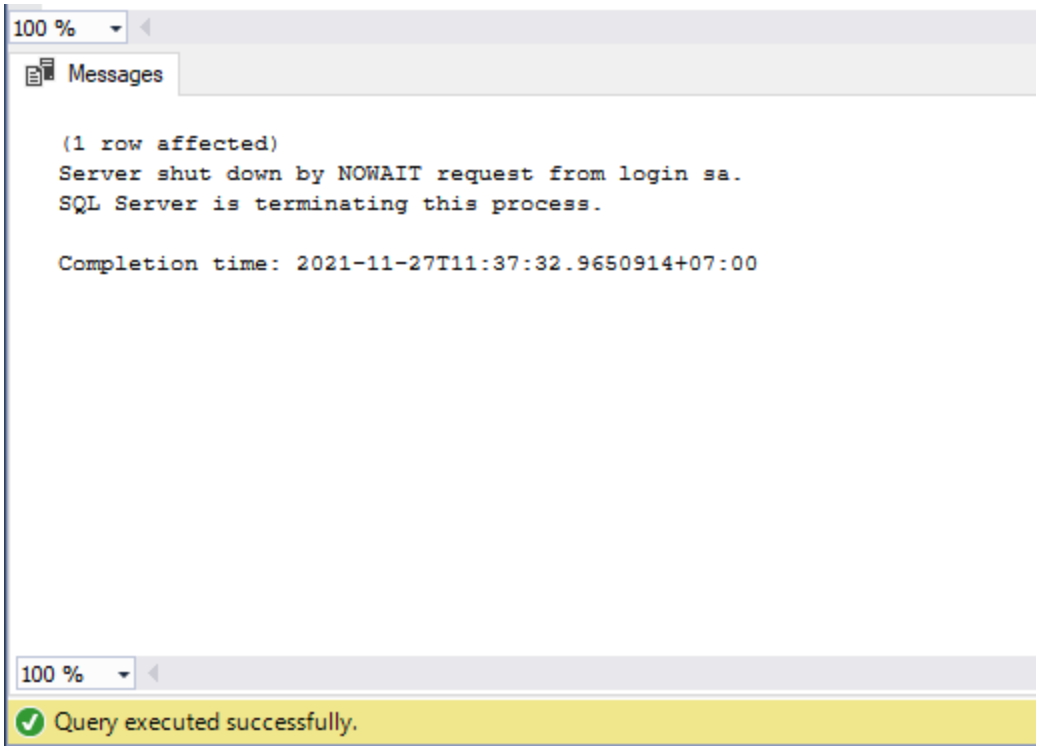
Results | Messages

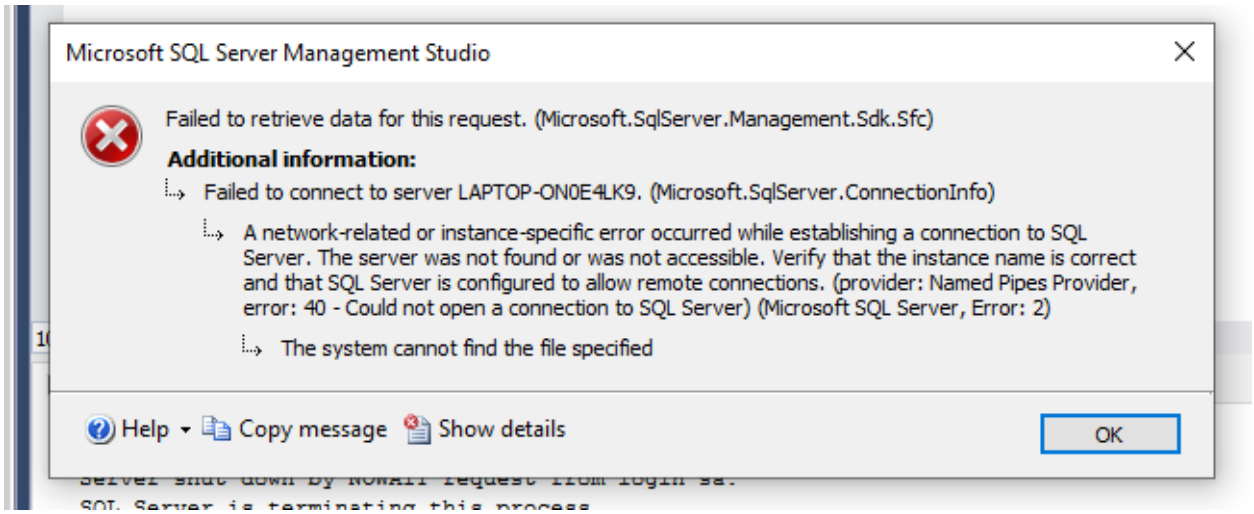
	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	10	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

T1 (where id=1 set price=15):

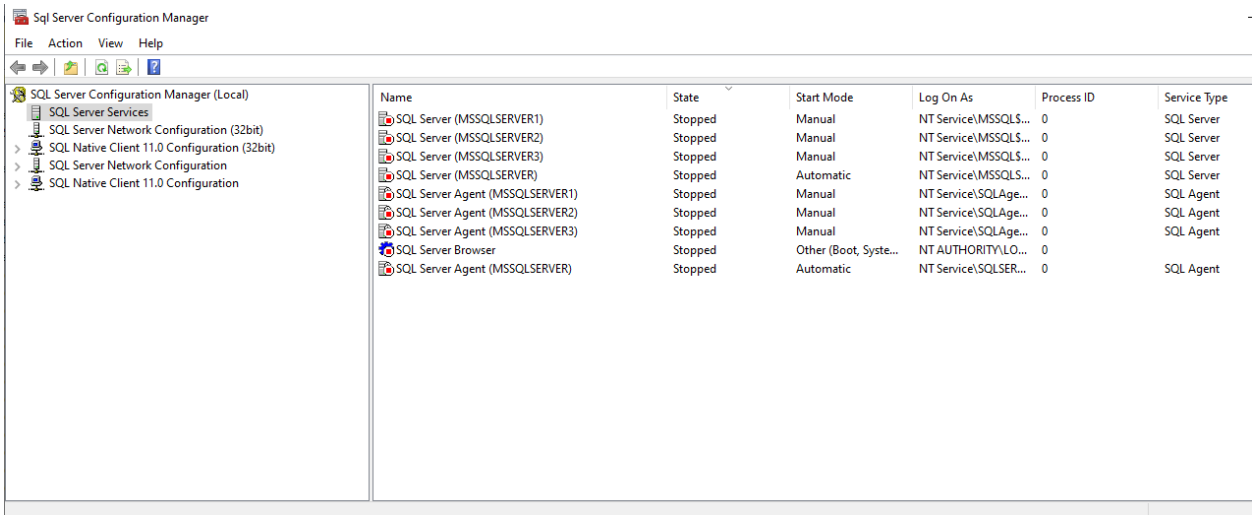
```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 15
where id = 1

commit transaction

shutdown with nowait
```

Run T1, the message goes like below, we also receive a pop-up dialog because our server is disconnected



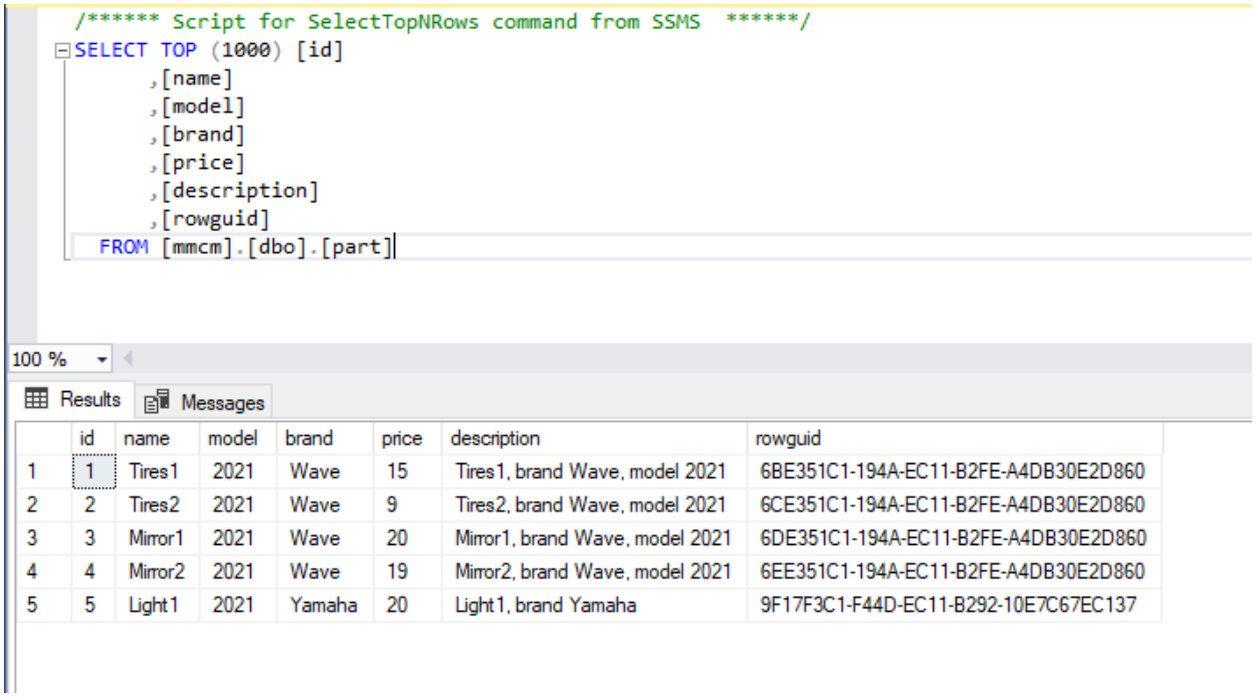


Our services are also stopped immediately after running T1:



Now let's reconnect the server and see if the change is made.

Run the 'select table part' query, the change is made in this site successfully:



Let's check the change in our master server, we can see the change is committed successfully:

```sql
set transaction isolation level serializable
begin transaction
SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]
commit transaction
```

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	15	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

Conclusion: if a site crashes, the committed transactions still apply successfully

2. Correct recovery before a commit: transaction must be aborted at all sites.

We still test our Part table

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	15	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

T1 (where id=1 set price=10):

```sql
set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 10
where id = 1

shutdown with nowait
```

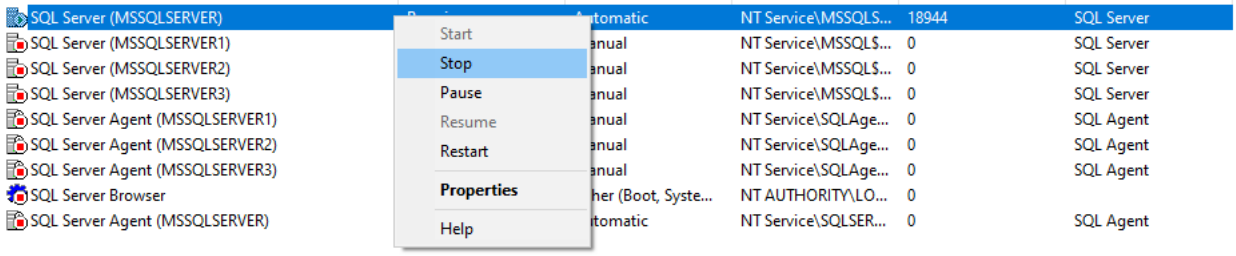
This time we won't commit T1 and demonstrate the server crash

```
100 %
Messages

(1 row affected)
The SHUTDOWN statement cannot be executed within a transaction or by a stored procedure.
Msg 0, Level 11, State 0, Line 0
A severe error occurred on the current command.  The results, if any, should be discarded.

Completion time: 2021-11-27T11:52:08.2291443+07:00
```

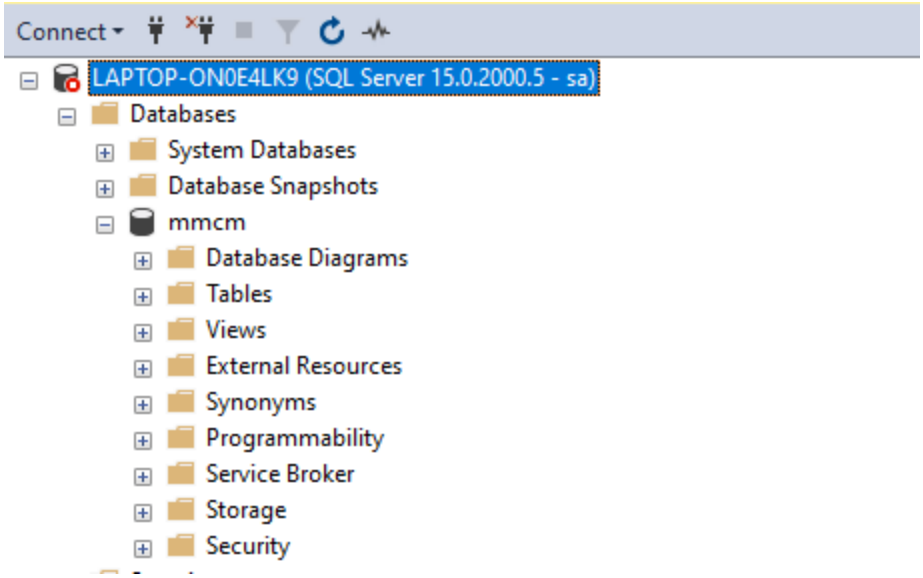
SQL Server won't allow to shutdown when running a uncommitted transaction, then we will change a little bit in our T1 statement:

```
set transaction isolation level serializable
begin transaction

update mmcm.dbo.part
set price = 10
where id = 1
```

We will stop the server manually (stop all the corresponding services):



Out SSMS turns out like below:



Now let's rerun it and see the Part table again (nothing's changed):

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]
```

100 %

Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	15	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

We also check the Part table from the master server (nothing's changed):

```sql
set transaction isolation level serializable
begin transaction
SELECT TOP (1000) [id]
      ,[name]
      ,[model]
      ,[brand]
      ,[price]
      ,[description]
      ,[rowguid]
  FROM [mmcm].[dbo].[part]
commit transaction
```

161 %

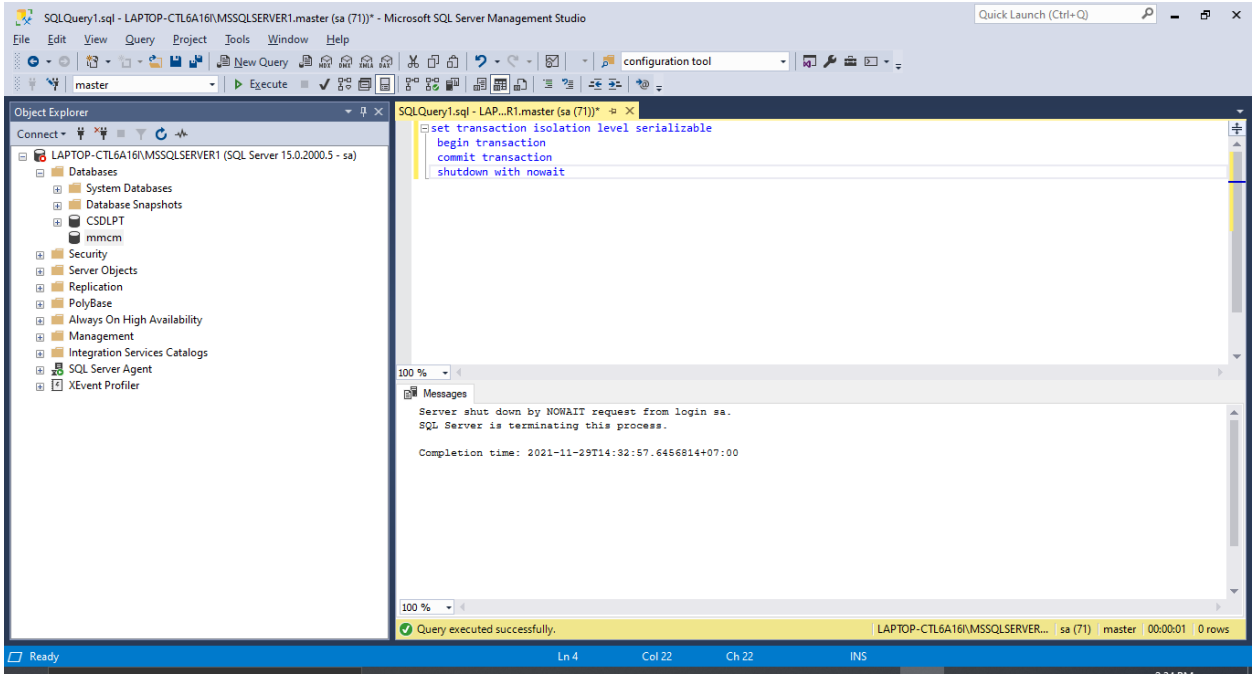
Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Tires1	2021	Wave	15	Tires1, brand Wave, model 2021	6BE351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Tires2	2021	Wave	9	Tires2, brand Wave, model 2021	6CE351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Mirror1	2021	Wave	20	Mirror1, brand Wave, model 2021	6DE351C1-194A-EC11-B2FE-A4DB30E2D860
4	4	Mirror2	2021	Wave	19	Mirror2, brand Wave, model 2021	6EE351C1-194A-EC11-B2FE-A4DB30E2D860
5	5	Light1	2021	Yamaha	20	Light1, brand Yamaha	9F17F3C1-F44D-EC11-B292-10E7C67EC137

Conclusion: 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 server.

3. **At least some situations where failure is non-blocking (remaining sites can run transactions that do not involve the failed sites.) Your system does not need to be non-blocking in all cases, but you should document failures that can result in blocking or conditions that can result in an inconsistent database (hopefully you will have none of the above.)**

First we will shut down the Hanoi branch



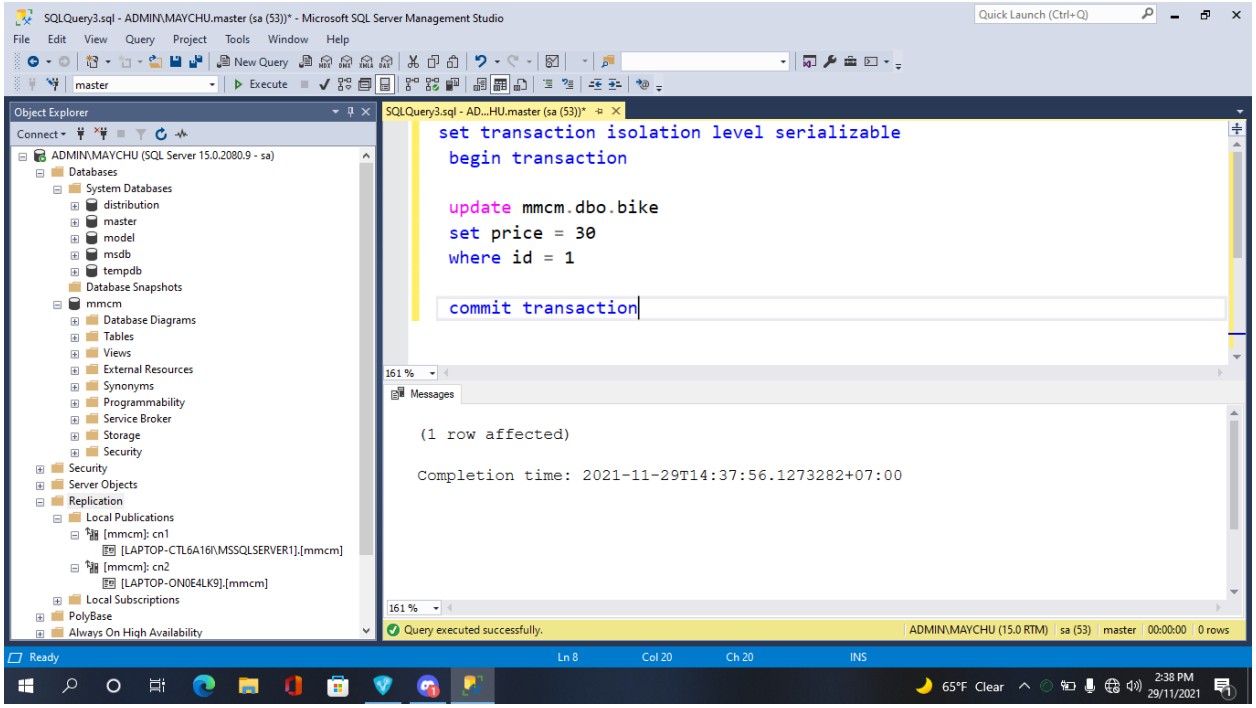
Now we will try to send transaction between other servers, suppose the master server update the `bike` table like below:

Before the transaction:

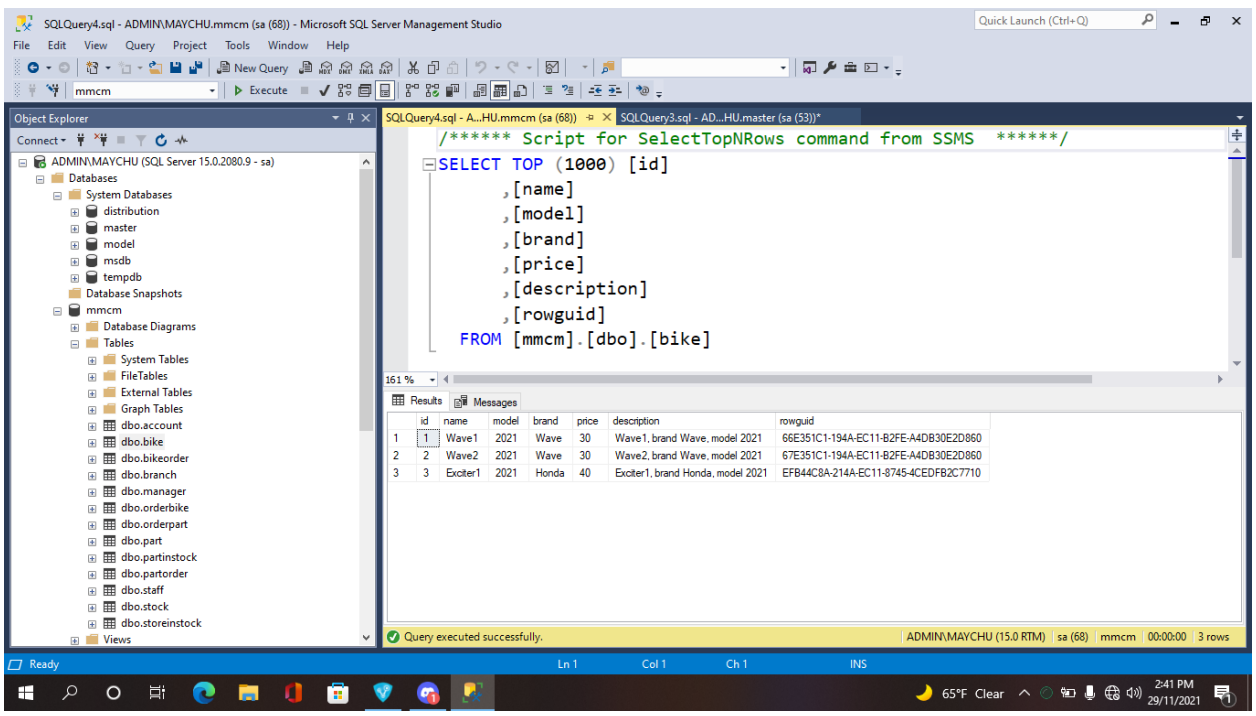
Results | Messages

	id	name	model	brand	price	description	rowguid
1	1	Wave1	2021	Wave	25	Wave1, brand Wave, model 2021	66E351C1-194A-EC11-B2FE-A4DB30E2D860
2	2	Wave2	2021	Wave	30	Wave2, brand Wave, model 2021	67E351C1-194A-EC11-B2FE-A4DB30E2D860
3	3	Exciter1	2021	Honda	40	Exciter1, brand Honda, model 2021	EFB44C8A-214A-EC11-8745-4CEDFB2C7710

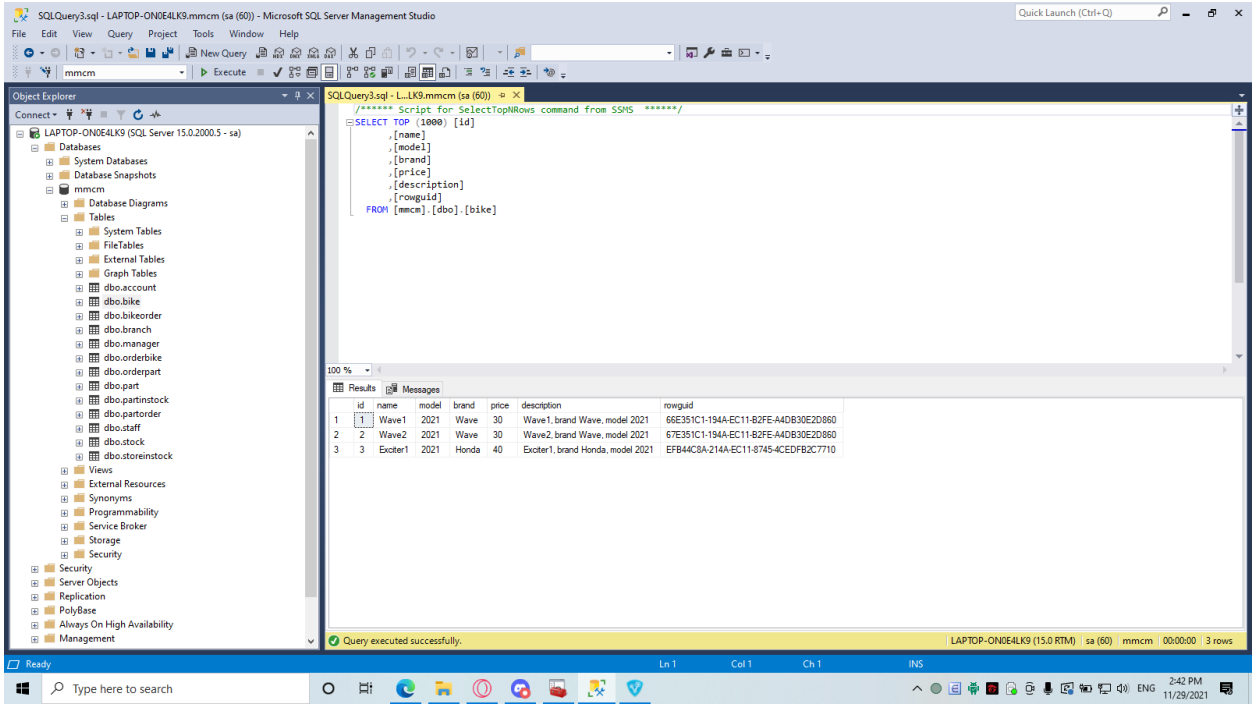
Run the update query (where id=1 update price=30):



The table `bike` in the master server after the transaction:



The table `bike` in the Danang server does apply the exact same transaction either (where id=1 the price now is 30 instead of 25):



Conclusion: As can be seen, 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.

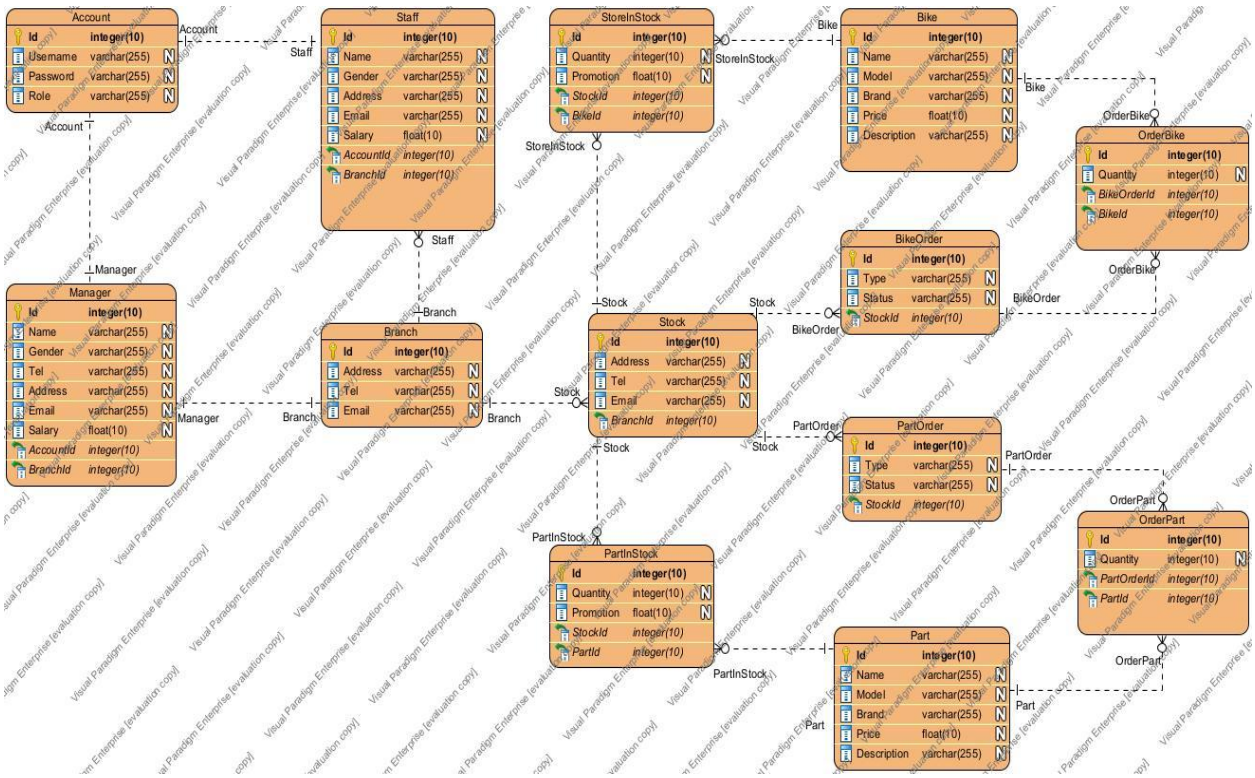
User manual (how-to-use)

Data

The distributed database system of the motorbike manufacturing company can be used to store essential data of the company. Such as:

- Informations of bikes and parts of bike in stocks
- Orders for importing and exporting bikes and parts
- Informations of different branches, stocks in each branch.
- Informations of branch managers and staffs
- Accounts of system users, could be either managers or staffs

The data are designed and stored in tables, the table details and relationships are described in the ERD below:



The data in these tables are distributed in different server sites (locations)

- 3 main sites based on locations: **Site1 - Hanoi**, **Site2 - Danang**, **Site 3 - Hochiminh**
- The only 1 site **(Site4)** which contains all information for all locations

Functions

Read/import/export/update data from the tables

Applications

These following applications could run on this database system:

1. List of staffs who have salary <500$
2. List of staffs who have salary >=500$ and <750$
3. List of staffs who have salary >= 750$
4. List of staffs in Hanoi's branch
5. List of staffs in Danang's branch
6. List of staffs in Hochiminh's branch
7. List of staffs that are not in Hanoi, Danang and Ho Chi Minh branch

8. List of managers in Hanoi's branch
9. List of managers in Danang's branch
10. List of managers in Hochiminh's branch
11. List of managers that are not in Hanoi, Danang and Ho Chi Minh branch
12. List of stocks in Hanoi's branch
13. List of stocks in Danang's branch
14. List of stocks in Hochiminh's branch
15. List of stocks that are not in Hanoi, Danang and Ho Chi Minh branch
16. List of bikes in Hanoi's stock
17. List of bikes in Danang's stock
18. List of bikes in Hochiminh's stock
19. List of bikes that are not in Hanoi, Danang and Ho Chi Minh branch
20. List of Parts in Hanoi branch stocks
21. List of Parts in Danang branch stocks
22. List of Parts in Ho Chi Minh branch stocks
23. List of Parts in not in Hanoi, Danang and Ho Chi Minh branch stocks
24. List of bike orders in stocks of Hanoi
25. List of bike orders in stocks of Danang
26. List of bike orders in stocks of Ho Chi Minh
27. List of bike orders that are not in Hanoi, Danang and Ho Chi Minh stocks
28. List of bikes in each Order in Hanoi branch stocks
29. List of bikes in each Order in Danang branch stocks
30. List of bikes in each Order in Hochiminh branch stocks
31. List of bikes in each Order that are not in Hanoi, Danang and Ho Chi Minh
32. List of part orders in Hanoi branch stocks
33. List of part orders in Danang branch stocks
34. List of part orders in Ho Chi Minh branch stocks
35. List of part orders that are not in Hanoi, Danang and Ho Chi Minh
36. List of Parts in each Order of Hanoi's Stocks
37. List of Parts in each Order of Danang's Stocks
38. List of Parts in each Order of Ho Chi Minh's Stocks
39. List of Parts in each Order that are not from Hanoi, Danang and Hochiminh stocks

How to use

Prerequisites: understand SQL, how to write queries, basic understanding of distributed systems.
Simply follow the instructions above and find which matches best your needs.

References

Commit protocols:
Distributed DBMS - Commit Protocols
Example: How SQL Server Explicitly Initiates a Transaction

Isolation levels:
Transaction Isolation Levels in DBMS
Transaction Isolation Levels (ODBC) - ODBC API Reference
What is the default transaction isolation level for SQL Server with ADO.NET?
locked What is Default Locking level in SQL Server 2008 or 2008 R2? RRS feed

Concurrency control:
Lock Based Concurrency Control Protocol in DBMS
DBMS Concurrency Control: Timestamp & Lock-Based Protocols
Two-phase locking
Two Phase Locking Protocol
Categories of Two Phase Locking (Strict, Rigorous & Conservative)
SQL Server's Locking Explained
3 Different Types of Read Write Conflict in DBMS [Explained with Example]

Failure/recovery:
Why recovery is needed in DBMS
Database Recovery Techniques in DBMS