A felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása a Pannon Egyetemen

EFOP-3.4.3-16-2016-00009

Advanced database management systems

Course notes

István Vassányi, PhD

vassanyi at almos dot uni-pannon dot hu

University of Pannonia, Department of Electrical Engineering and Information Systems

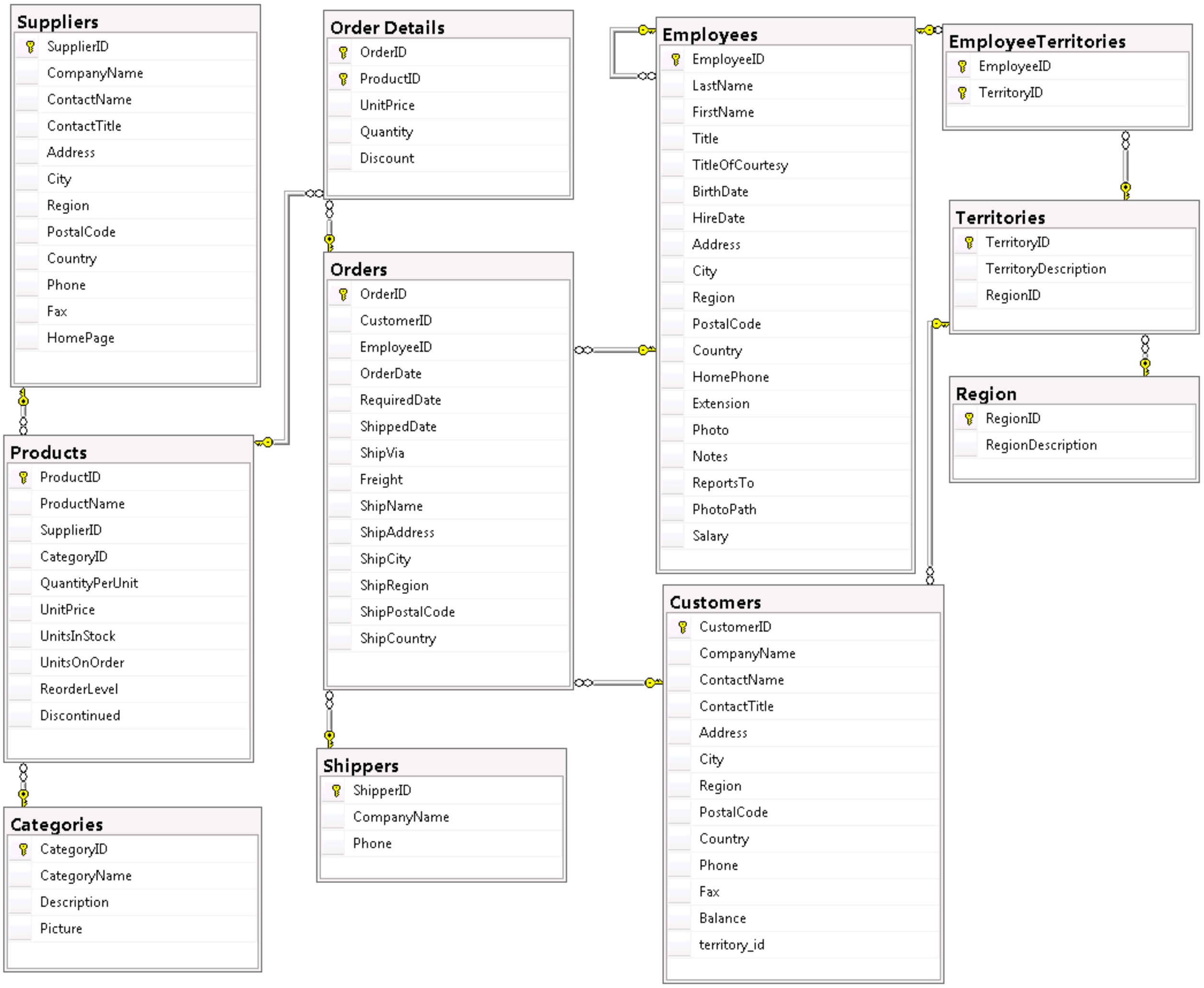
Veszprém, Hungary

2018-2022

CONTENTS

1. Review of core database skills

Welcome. In some of the demos in this course, we'll use the Northwind sample relational database[1] and the SQL server 2022 technology from Microsoft. The Northwind database was designed to support a small company trading with consumables. It includes an inventory and tables for the administration of the orders. The table and file names should be self-explanatory.



Modeling

- First we review the core relational modeling concepts for On-Line Transaction Processing (OLTP) databases, demonstrated on the Northwind database: Customers, Employees, Orders, OrderDetails, Products, Categories, Territories tables.
    - We start with a conceptual model (domain model or entity relationship model) that we derive from the use cases and our aim is to develop the logical database model

[1] You can download the database dump from https://www.microsoft.com/en-us/download/details.aspx?id=23654 In this course, we modified the original database by adding a foreign key territory_id to the Customers table and an extra field Salary to the Employees table, for the sake of some exercises.

- The relational model is the most widely used paradigm to support traditional business processes due to its simplicity
- Entities, attributes, instances, identifiers are implemented in the relational model as tables, fields, records, primary keys. Keys may be composed from multiple fields
- Only one value in any single cell—no redundancy and no inconsistency is allowed in third normal form (3NF). Characteristics of 3NF:
    - Each table has a primary key that may be composed of multiple fields, and on which all the other fields functionally depend;
    - In case of composite (multi-field) keys, all of the non-key fields depend on the whole key, and not just a part of it i.e. there are no partial dependencies;
    - The non-key fields depend on no other field(s) except the key, i.e. there are no transitive dependencies within a table.
- All tables are connected
- 1:N (one-to-many) relationships are implemented with foreign keys (e.g. Orders.EmployeeID)
- N:M (many-to-many) relationships are implemented with linking tables (e.g. EmployeeTerritories)
- 1:1 (one-to-one) relationship is not exemplified in the Northwind database
    - A normal 1:1 relationship could be a CompanyCar table if an employee may have at most one company car allocated
    - A specialization type 1:1 relationship could be an ExciseProducts table for excise goods with extra fields ExciseDutyAmount, RegBarCode etc.
- Linking tables usually have composite keys. We generate keys only if an external reference is needed.
- The relationship structure of an OLTP schema reveals the key transactions of the application that uses the database.
    - Snowflake or snowball structure, each snowflake supporting one or more transactions.
    - Base tables are at the leaves (e.g. Region, Customers, Categories)
    - Transactional tables or event-tables are in the middle (Order Details, EmployeeTerritories). These tables form the ‘beating heart’ of the information system.

Feature	**Base Tables**	**Transactional Tables**
Position in the schema	Leaf. Does not reference any other table	Centre. References directly or indirectly all tables
Size	Small	Large

Speed of change	Slow. Cold backups may be sufficient.	Fast. Hot backups are needed.

- Connection between a properly designed Graphical User Interface (GUI) and the relational schema
  - Hidden or read-only label: key
  - Editable text boxes: attributes (fields) that depend on the key
  - Dropdown/combo lists: references to base tables
  - Checkbox with an additional text box: specialization
  - Dropdown tables or lists: 1:N relationships
- Further reading on modeling:
  - https://www.safaribooksonline.com/library/view/relational-theory-for/9781449365431/ch01.html
  - http://www.blackwasp.co.uk/RelationalDBConcepts.aspx
  - https://www.tutorialspoint.com/ms_sql_server/index.htm
- PRACTICE: create and extend the sample database
  - Install MS SQL Server 2016 or later, start the database service and connect to it using MS Management Studio.
  - Run the northwind database create dump and review the tables with the GUI tools
  - Draw a logical database model diagram similar to the diagram above
  - Add the fields Employees.Salary and Customers.territory_id
  - Design and implement an extension to the database to model the following scenario. *We send our employees to regular training sessions where they learn various skills. Training sessions are organized by contracted third party companies. We have a list of required skills (like "grade B business presentation" or "accounting basics" etc.) for each employment category (like "sales manager", see Employees.Title) that they must learn within 10 years after the beginning of their employment. For each training, we store the duration (beginning and ending date), location, organizing company, skills taught, participants, their training status (like "enrolled", "started", "completed", "aborted") and their exam results separately for the various skills. With respect to the companies organizing the trainings, we store the fees paid by our company for the training sessions each year.*
  - (Add the new tables to the database diagram and enter some test data)
  - SOLUTION: train_tables.sql[2]

[2] For the solutions of the students' test problems please contact the author

Querying

- We review the basics of Structured Query Language (SQL) querying like selecting, grouping, joining. Example queries:
  - Value of each order
  - Minimum and maximum quantities sold for each product on a yearly basis
  - Which employee sold the most pieces of the most popular product in 1998?

```
--      Value of each order
select o.orderid, o.orderdate,
    str(sum((1-discount)*unitprice*quantity), 15, 2) as order_value,
    sum(quantity) as no_of_pieces,
    count(d.orderid) as no_of_items
from orders o inner join [order details] d on o.orderid=d.orderid
group by o.orderid, o.orderdate
order by sum((1-discount)*unitprice*quantity) desc

--      Quantities sold for each product on a yearly basis
select p.ProductID, p.ProductName, year(o.orderdate), SUM(quantity) as quantity
from orders o inner join [order details] d on o.orderid=d.orderid
inner join Products p on p.ProductID=d.ProductID
group by p.ProductID, p.ProductName, year(o.orderdate)
order by p.ProductName

--      Which employee sold the most pieces of the most popular product in 1998?
select top 1 u.titleofcourtesy+' '+u.lastname+' '+ u.firstname +' ('+u.title +')'  as name,
    sum(quantity) as pieces_sold,
    pr.productname as productname
from orders o inner join [order details] d on o.orderid=d.orderid
    inner join employees u on u.employeeid=o.employeeid
    inner join products pr on pr.productid=d.productid
where year(o.orderdate)=1998 and d.productid =
    (select top 1 p.productid
    from products p left outer join [order details] d on p.productid=d.productid
    group by p.productid
    order by count(*) desc)
group by u.employeeid, u.titleofcourtesy, u.title, u.lastname, u.firstname,
pr.ProductID,pr.productname
order by sum(quantity) desc
```

- For more examples and a systematic overview of SQL querying, see the Appendix
- Further reading on querying:
  - https://docs.microsoft.com/en-us/sql/t-sql/queries/queries
- PRACTICE: using the tables implemented in the first practice, implement the following queries
  - What are the missing skills for Mrs. Peacock?
  - Are there any sessions in the future that are still required for Peacock to attend?
  - What is the first and last training date and the average duration of trainings in days?
  - Which employee has the most skills with an exam result above ‘fail’?
  - What is the total fee paid for all training sessions in which our most skilled employee (see above) participated?

    - Which required skill(s) have not yet been addressed by any training session?
    - SOLUTION: train_solution.sql

Programming

- Besides SQL, procedural transactional logic can be implemented in the scripting language T-SQL, and it can be run and stored on the server side
    - Pros and cons for server side business logic
        - ✓ Simple architecture
        - ✓ Technological neutrality
        - ✓ Data safety
        - ✓ Manageability
        - ✓ Efficiency
        - ✓ Readable code
        - ✗ Low level
        - ✗ Poor software technological support
        - ✗ Expensive scalability
    - The bottom line is that the part of business logic that involves simple, set-based operations on *large volumes* of *structured* data are best implemented and managed on the database server in the form of stored procedures, functions, triggers and jobs. Procedurally sophisticated parts of the business logic that call for a high level, object-oriented programming environment, should be implemented on an application server.
    - The elements of server side programmability
        - Special SQL keywords for control flow: DECLARE, SET, BEGIN/END, IF/ELSE, WHILE/BREAK/CONTINUE, RETURN, WAITFOR/DELAY/TIME, GOTO
        - Error handling: TRY/CATCH/THROW/RAISERROR
        - Objects supporting programmability: CREATE PROCEDURE/FUNCTION/TRIGGER
        - Transactional support: BEGIN/COMMIT/ROLLBACK TRANSACTION
    - Below is a simple example of a T-SQL script and its stored procedure equivalent. The similar user defined function can be used in a SELECT statement.

```sql
--a simple script that demonstrates the elements of T-SQL
--we search for an emplyee, and if we find a single matching record,
--we increase the salary of the employee by 10%
set nocount on
declare @name nvarchar(20), @address nvarchar(max), @res_no int, @emp_id int
set @name='Fuller'
select @res_no=count(*) from Employees where LastName like @name + '%'
if @res_no=0 print 'No matching record.'
else if @res_no>1 print 'More than one matching record.'
else begin  --a single hit
       select @address=Country+', '+City+' '+Address, @emp_id=EmployeeID
              from Employees where LastName like @name
```

```
        print 'Employee ID: ' + cast(@emp_id as varchar(10)) + ', address: ' + @address
        update Employees set salary=1.1*salary where EmployeeID=@emp_id
        print 'Salary increased.'
end
go

--wrap it in a stored procedure
create procedure sp_increase_salary @name nvarchar(40)
as
set nocount on
declare @address nvarchar(max), @res_no int, @emp_id int
select @res_no=count(*) from Employees where LastName like @name + '%'
if @res_no=0 print 'No matching record.'
else if @res_no>1 print 'More than one matching record.'
else begin  --a single hit
        select @address=Country+', '+City+' '+Address, @emp_id=EmployeeID
                from Employees where LastName like @name
        print 'Employee ID: ' + cast(@emp_id as varchar(10)) + ', address: ' + @address
        update Employees set salary=1.1*salary where EmployeeID=@emp_id
        print 'Salary increased.'
end
go
--test
select Salary from Employees where LastName like 'Fuller%'
exec sp_increase_salary 'Fuller'
select Salary from Employees where LastName like 'Fuller%'

--a scalar valued function that returns the salary of a person or 0 if the person is not found
go
create function fn_salary (@name nvarchar(40)) returns money as
begin
        declare @salary money, @res_no int
        select @res_no=count(*) from Employees where LastName like @name + '%'
        if @res_no <> 1 set @salary=0
        else select @salary=Salary from Employees where LastName like @name  + '%'
        return @salary
end
go
--test
select [your user name].fn_salary('Fuller') as salary
```

- Note that a **stored procedure** can return multiple record sets if it contains multiple SELECT statements without variable assignment. Parameters passed by value as shown in the example above are INPUT type parameters. Stored procedures may also return scalar values in OUTPUT parameters (not shown in the example). Stored procedures may also call other stored procedures or functions, therefore they can be used to implement complex business logic on the DB server.

- A **user defined function** differs from a stored procedure in that it must have a single return value, the type of which may be scalar (like money), or table. The last statement of a function must be a RETURN. The advantage of user defined functions over stored procedures is that a function may be called from inside a SELECT statement like any other built-in SQL function like DATEDIFF etc., thus it can add a lot to the flexibility of static SQL queries.

- PRACTICE

    - Using the training queries, create a stored procedure that returns the missing skills for an employee name passed as a parameter. The stored procedure should return a table with a single field containing the missing skills. If the employee cannot be identified, return an error message and no table.

- ○ Using the training queries, create a table-valued function that returns the missing skills for an employeeID, in the form of a table. Hint: use ‘returns table’ in the function specification.

- In order to demonstrate a more realistic business process, here is an example script for making a new Northwind order that contains a single order item. The scenario is that the company office receives an urgent order from a valued customer over the phone. Such a process is a typical business transaction.

```
--variables
declare @prod_name varchar(20), @quantity int, @cust_id nchar(5) --we receive the textual
customer id over the phone
declare @status_message nvarchar(100),  @status int --the result of the business process
declare @res_no int --No of hits
declare @prod_id int, @order_id int --IDs
declare @stock int --existing product stock
declare @cust_balance money --customers balance
declare @unitprice money --unit price of product

-- parameters
set @prod_name = 'boston'
set @quantity = 10
set @cust_id = 'AROUT'

begin try
        select @res_no = count(*) from products where productname like '%' + @prod_name + '%'
        if @res_no <> 1 begin
                set @status = 1
                set @status_message = 'ERROR: Ambiguous Product name.';
        end else begin
                -- if we find a single product, we look for the key and the stock
                select @prod_id = productID, @stock = unitsInStock from products where
productName like '%' + @prod_name + '%'
                -- is the stock sufficient?
                if @stock < @quantity begin
                        set @status = 2
```

```
                        set @status_message = 'ERROR: Stock is insufficient.'
                end else begin
                -- Does the customer have credit?
                        select @cust_balance = balance from customers where customerid =
@cust_id
                                                --if there is no hit, the @cust_balance is null
                                                --there cannot be more than one hit
                        select @unitprice = unitPrice from products where productID = @prod_id -
-no discount
                        if @cust_balance < @quantity*@unitprice or @cust_balance is null begin
                                set @status = 3
                                set @status_message = 'ERROR: Customer not found or balance
insufficient.'
                        end else begin
                -- no more checks, we start the transaction (3 steps)
                -- 1. decrease the balance
                        update customers set balance = balance-(@quantity*@unitprice) where
customerid=@cust_id
                -- 2. new record in the  Orders, Order Details
```

```
                                insert into orders (customerID, orderdate) values (@cust_id,
getdate()) --orderid: identity
                                set @order_id = @@identity  --result of the last identity insert
                                insert [order details] (orderid, productid, quantity, UnitPrice)
--here we make an error
                                        values(@order_id, @prod_id, @quantity, @unitprice) --here
we make an error
--                              insert [order details] (orderid, productid, quantity, UnitPrice,
Discount) --the correct line
--                                      values(@order_id, @prod_id, @quantity, @unitprice, 0) --
the correct line
                -- 3. update product stock
                                update products set unitsInStock = unitsInStock - @quantity where
productid = @prod_id
                                set @status = 0
                                set @status_message = cast(@order_id as varchar(20)) + ' order
processed successfully.'
                        end
                end
        end
        print @status
        print @status_message
end try
begin catch
        print 'OTHER ERROR: '+ ERROR_MESSAGE() + ' (' + cast(ERROR_NUMBER() as varchar(20)) +
')'
end catch
go

--we set parameters for testing
set nocount off
update products set unitsInStock = 900 where productid=40
update customers set balance=1000 where CustomerID='AROUT'
delete [Order Details] where OrderID in (select orderid from Orders where CustomerID='AROUT'
and EmployeeID is null)
delete Orders where CustomerID='AROUT' and EmployeeID is null
--we run the script and then check:
select * from Customers where CustomerID='AROUT'
select * from Products where productid=40
select top 3 * from Orders where CustomerID='arout' order by OrderDate desc

--Seems fine. However we neglected a NOT NULL constraint of the discount field:
--"OTHER ERROR: Cannot insert the value NULL into column 'Discount'"
--Even worse, we still decreased the balance of the customer!

--in a concurrent environment, other errors may manifest as well

--after correction, test the other two branches as well
```

- Further reading on programming:
  - https://docs.microsoft.com/en-us/sql/t-sql/language-elements/control-of-flow
- PRACTICE: using the tables and scripts implemented in the previous practices,
  - Write a script that checks whether an employee needs any of the skills offered by a training session, and if yes, **enroll the employee** for all such sessions.
  - Run the script in a stored procedure.

- SOLUTION: train_solution.sql

Cursors

Cursors can be used for problems for which the procedural row-by-row approach is more suitable than the set-based querying approach.

EXAMPLE for cursor syntax

```
declare @emp_id int, @emp_name nvarchar(50), @i int, @address nvarchar(60)
declare cursor_emp cursor for
    select employeeid, lastname, address from employees order by lastname
set @i=1
open cursor_emp
fetch next from cursor_emp into @emp_id, @emp_name, @address
while @@fetch_status = 0
begin
    print cast(@i as varchar(5)) + ' EMPLOYEE:'
    print 'ID: ' + cast(@emp_id as varchar(5)) + ', LASTNAME: ' + @emp_name + ', ADDRESS: ' + 
@address
    set @i=@i+1
    fetch next from cursor_emp into @emp_id, @emp_name, @address
end
close cursor_emp
deallocate cursor_emp
go
--equivalent to this with a SELECT
select 'ID: ' + cast(employeeid as varchar(5)) + isnull(', LASTNAME: ' + lastname, '') + 
isnull( ', ADDRESS: ' + address, '')
from employees order by lastname
--or, with a row number
select cast(row_number() over(order by lastname) as varchar(50))+
'. ügynök: ID: ' + cast(employeeid as varchar(5)) + isnull(', LASTNAME: ' + lastname, '') + 
isnull( ', ADDRESS: ' + address, '')
from employees
```

PRACTICE: Implement a cursor that iterates the USA customers and prints the number of their respective orders row by row.

Transaction management

- The core transactional concepts
  - We define ‘**transaction**’ as a logically coherent sequence of operations in a business process. ‘Logically coherent’ means that the operations form a semantic unit. Transactions may be **nested** e.g. the transaction of buying a helicopter includes the transaction of the customer identifying herself and the transaction of paying the bill by bank transfer etc.
  - **Atomicity**, **consistency**, **isolation** and **durability** requirements for environments implementing transactions. We violated the atomicity and isolation requirement in our last order processing example.
  - There are **implicit** and **explicit** (programmed) **transactions**. Implicit transactions are all SQL DML statements.
  - Transactions in T-SQL are programmed with the BEGIN/COMMIT/ROLLBACK TRANSACTION statements. The transaction consists of all statements between the BEGIN TRANSACTION and a COMMIT TRANSACTION or ROLLBACK TRANSACTION statement.

COMMIT closes the transaction and frees all the resources like table locks etc. that were used by the server for transaction management. ROLLBACK does the same after undoing all changes performed by all the statements of the transaction. For this to be possible, the server uses a sophisticated logging mechanism called the **Write-Ahead Log** (WAL). If not truncated or backed up, the transactional log may grow bigger than the database itself.

- In MS SQL Server, if XACT_ABORT is ON and one of the transaction's statements causes an error, the server stops executing the transaction and performs an automatic ROLLBACK.

EXAMPLE

```
--simple demo for atomicity, with xact_abort on
set xact_abort off
delete t2
go
begin tran
        insert t2 (id, t1_id) values (10, 1)
        insert t2 (id, t1_id) values (11, 2) --foreign key constraint violation
        insert t2 (id, t1_id) values (12, 3)
commit tran
go
--"The INSERT statement conflicted with the FOREIGN KEY constraint ..." etc
select * from t2
id      t1_id
10      1
12      3
--atomicity was not preserved
set xact_abort on
delete t2
go
begin tran
        insert t2 (id, t1_id) values (10, 1)
        insert t2 (id, t1_id) values (11, 2) --foreign key constraint violation
        insert t2 (id, t1_id) values (12, 3)
commit tran
go
--"The INSERT statement conflicted with the FOREIGN KEY constraint ..." etc
select * from t2
id      t1_id
--atomicity was preserved
```

- **Nested transactions** technically mean multiple BEGIN TRANSACTION statements. A single ROLLBACK will roll back all transactions that have been begun, see example below

```
begin tran
        print @@trancount  --1
        begin tran
                print @@trancount  --2
        commit tran
        print @@trancount  --1
commit tran
print @@trancount  --0

begin tran
        print @@trancount  --1
        begin tran
                print @@trancount  --2
rollback tran
print @@trancount  --0
```

- It is a serious **programming error** not to close a transaction by either a COMMIT or a ROLLBACK. An unterminated transaction will continue consuming server resources and will eventually cripple the system. The **@@TRANCOUNT** global variable may be used to check whether the current connection has an unterminated transaction.

EXAMPLE: In order to correct the shortcomings of the example order processing script, we wrap it into a stored procedure, and add TRY/CATCH error handling and transactional support.

```
go
create procedure sp_new_order
@prod_name nvarchar(40), @quantity smallint, @cust_id nchar(5)
as
set nocount on
set xact_abort on
--variables
declare @status_message nvarchar(100),  @status int --the result of the business process
declare @res_no int --No of hits
declare @prod_id int, @order_id int --IDs
declare @stock int --existing product stock
declare @cust_balance money --customers balance
declare @unitprice money --unit price of product
begin tran
begin try
        select @res_no = count(*) from products where productname like '%' + @prod_name + '%'
        if @res_no <> 1 begin
                set @status = 1
                set @status_message = 'ERROR: Ambiguous Product name.';
        end else begin
                -- if we find a single product, we look for the key and the stock
                select @prod_id = productID, @stock = unitsInStock from products where
productName like '%' + @prod_name + '%'
                -- is the stock sufficient?
                if @stock < @quantity begin
                        set @status = 2
                        set @status_message = 'ERROR: Stock is insufficient.'
                end else begin
                -- Does the customer have credit?
                        select @cust_balance = balance from customers where customerid =
@cust_id
                                                --if there is no hit, the @cust_balance is null
                                                --there cannot be more than one hit
                        select @unitprice = unitPrice from products where productID = @prod_id -
-no discount
                        if @cust_balance < @quantity*@unitprice or @cust_balance is null begin
                                set @status = 3
                                set @status_message = 'ERROR: Customer not found or balance
insufficient.'
                        end else begin
                -- no more checks, we start the transaction (2 steps)
                -- 1. decrease the balance
                        print 'Processing order...'
                                update customers set balance = balance-(@quantity*@unitprice)
where customerid=@cust_id
                -- 2. new record in the  Orders, Order Details
                                insert into orders (customerID, orderdate) values (@cust_id,
getdate()) --orderid: identity
                                set @order_id = @@identity  --result of the last identity insert
                                insert [order details] (orderid, productid, quantity, UnitPrice)
values(@order_id, @prod_id, @quantity, @unitprice) --here we make an error
```

```
        --        insert [order details] (orderid, productid, quantity, UnitPrice, 
Discount) values(@order_id, @prod_id, @quantity, @unitprice, 0) --the correct line
                set @status = 0
                set @status_message = 'Order No. ' + cast(@order_id as 
varchar(20)) + ' processed successfully.'
            end
        end
    end
    print 'Status: ' + cast(@status as varchar(50))
    print @status_message
    if @status = 0 commit tran else begin
        print 'Rolling back transaction'
        rollback tran
    end
end try
begin catch
    print 'OTHER ERROR: '+ ERROR_MESSAGE() + ' (' + cast(ERROR_NUMBER() as varchar(20)) + 
')'
    print 'Rolling back transaction'
    rollback tran
end catch
go

--test
--we set parameters for testing
set nocount off
update customers set balance=1000 where CustomerID='AROUT'
delete [Order Details] where OrderID in (select orderid from Orders where CustomerID='AROUT' 
and EmployeeID is null)
delete Orders where CustomerID='AROUT' and EmployeeID is null
--we run the stored proc
exec sp_new_order 'boston', 10, 'Arout'
--check the results:
select * from Customers where CustomerID='AROUT' --should be 816
select top 3 * from Orders o inner join [Order Details] od on o.OrderID=od.OrderID
    where CustomerID='arout' order by OrderDate desc --should see the new item
select @@trancount --must be 0
```

  - Test the above stored procedure for various errors: programming errors and logical errors like insufficient stock. Check the integrity of the database. Check that the transactional support prevents any serious errors.

- In order to ensure isolation, the server uses locks on rows (records), ranges or tables. An **Isolation Level** is a locking strategy enforced by the server. The main lock types on MS SQL Server are Read (shared), Write (exclusive) and Update. Below are the 4 ANSI standard isolation levels, though current database technologies support more than just these 4.
  - READ UNCOMMITTED: no locking
  - READ COMMITTED: locks removed after the completion of the SQL statement
  - REPEATABLE READ: locks that were granted for the transaction are kept until the end of the transaction
  - SERIALIZABLE: other transactions cannot insert records into a table for which a transaction has a row or range lock, phantom read is not possible

```
--simple demo for isolation: the webshop case
create table test_product(id int primary key, prod_name varchar(50) not null, sold 
varchar(50), buyer varchar(50))
```

```
insert test_product(id, prod_name, sold) values (1, 'car', 'for sale')
insert test_product(id, prod_name, sold) values (2, 'horse', 'for sale')
go
select * from test_product
update test_product set sold='for sale', buyer=null where id=2
go
set tran isolation level read committed --the default
go
begin tran
declare @sold varchar(50)
select @sold=sold from test_product where id=2
if @sold='for sale' begin
    waitfor delay '00:00:10' --now we are performing the bank transfer
    update test_product set sold='sold', buyer='My name' where id=2
    print 'sold successfully'
end else print 'product not available'
commit tran
go
--we run the above transaction concurrently in two query editors
--the second script:
set tran isolation level read committed
go
begin tran
declare @sold varchar(50)
select @sold=sold from test_product where id=2
if @sold='for sale' begin
    waitfor delay '00:00:10' --now we are performing the bank transfer
    update test_product set sold='sold', buyer='Your name' where id=2 --note the diff
    print 'sold successfully'
end else print 'product not available'
commit tran
go
--check what happens:
select * from test_product
id      prod_name      sold           buyer
1       car                    for sale       NULL
2       horse          sold           Your name
--The horse was sold successfully to two customers, but only Your name will receive it. Very
awkward.
update test_product set sold='for sale', buyer=null where id=2
--Now try the same with set tran isolation level repeatable read
--"Transaction (Process ID 53) was deadlocked on lock resources with another process and has
been chosen as the deadlock victim. Rerun the transaction."
--No logical error. Only one horse is sold.

--Conclusion: be careful to select the right isolation level.
```

EXAMPLE of a dummy stored procedure syntax using a transaction:

```
go
create procedure sp_example (@emp_id int)
as
set xact_abort on --auto rollback in case of any error
begin tran
begin try
       declare @i int
       select @i=count(*) from employees where EmployeeID=@emp_id
       if @i>0 print 'Employee found: ' + cast(@emp_id as varchar(50))
       else print 'Not found: ' + cast(@emp_id as varchar(50))
```

```
        if @i>0 begin
                update employees set salary=salary*1.1 where EmployeeID=@emp_id
                commit tran
                print 'Salary successfully increased'
        end else begin
                print 'Rolling back transaction'
                rollback tran
        end
end try
begin catch
        print 'OTHER ERROR: '+ ERROR_MESSAGE() + ' (' + cast(ERROR_NUMBER() as
varchar(20)) + ')'
        print 'Rolling back transaction'
        rollback tran
end catch
go
--test
exec sp_example 12  --Not found: 12 Rolling back transaction
exec sp_example 11  --Employee found: 11 Salary successfully increased
```

- Further reading on transaction management:
    - https://docs.microsoft.com/en-us/sql/t-sql/language-elements/control-of-flow
    - https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver16
    - https://www.sqlshack.com/transactions-in-sql-server-for-beginners/
- PRACTICE: Add transactional support to your own training management stored procedure and test it for various errors.

2. Loose coupling based on triggers and jobs

Problem scenario

The new orders are stored in the Orders and Orderitems tables by a third party management and trading application that has no open API, or for any other reason refuses to generate service level events. Therefore, in the current order processing workflow at the Northwind Traders Ltd Co. the trading department communicates with the Shipping and Logistics (SL) division via email (or any other manual messaging system) about the new or changed orders. Our company is responsible for IT support in SL management. The head of SL division prepares the detailed daily work plans every morning for the various units according to the emails received from the trading department. For this, she uses our software tool. Both the trading and the SL use the same Northwind SQL Server database.

We are asked to relieve the trading and SL staff from manually writing emails and manually entering data from emails into another application by automating the order processing workflow as much as possible.

Solution

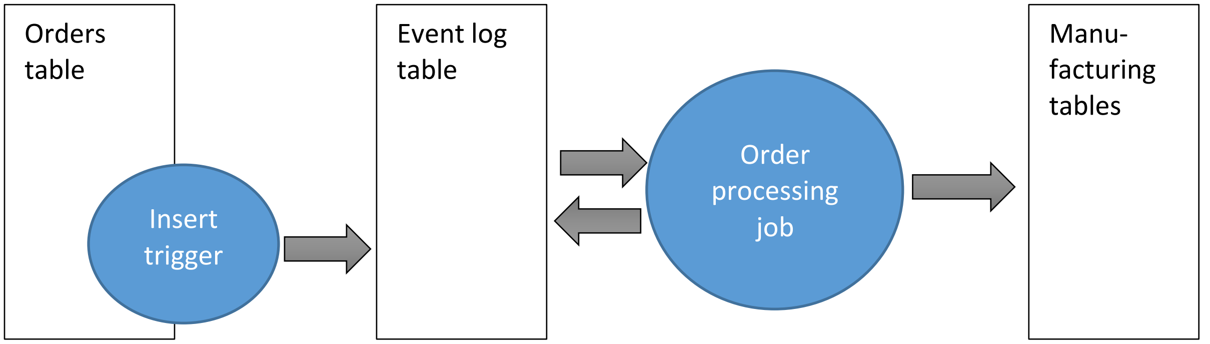
Since the trading system is a 'black box', we must rely on database level events. Every time an order is created or modified, we must run the required (rather complex) logic on the database that creates or changes the required records in the SL tables like Products. Thus both the writing and the processing of emails will be unnecessary.

It is, however, vital that our solution *should not at the least interfere* with the trading system. It cannot significantly slow down the order saving process, nor may any error that may occur while processing an order event on the SL side be propagated back to the trading system.

For this reason, we use the *loose coupling* concept. We only log the INSERT and UPDATE events on orders via a trigger in a special table, and process these events in batches executed by a scheduled job. The job also keeps track of the state and results of the processing of each event. Since the processing of the event is performed out-of-process, a processing error does not manifest as an error in the trading system.

*Note: a **trigger** is a special stored procedure that is invoked automatically by the database management system upon database events like table INSERT, UPDATE or DELETE.*

System overview:



A short overview on triggers

Triggers are special procedures stored on the server and run automatically when a pre-defined condition is satisfied. SQL Server supports the following types of triggers with respect to the trigger event:

- DML triggers (table level triggers) that are executed when a DELETE, INSERT or UPDATE action is performed on a table
- DDL triggers (database level triggers) that are fired when the schema of the database changes e.g. a table is created
- Logon triggers (server level triggers) that are fired after the authentication phase of logging in finishes

We focus now on DML triggers. The definition of the trigger includes the target table, the trigger event (DELETE, INSERT or UPDATE) and the mode of operation. SQL server supports the following operational modes:

- AFTER: fired after the successful execution of the specified SQL statement. This means that all eventual check and other constraints and cascade updates/deletes associated with the DML statement have executed successfully. We can place multiple triggers on the same object, even of the same type, like two INSERT triggers. In this case, the order of execution can be influenced by setting trigger properties.
- INSTEAD OF: the DML statement is not executed at all, only the trigger.

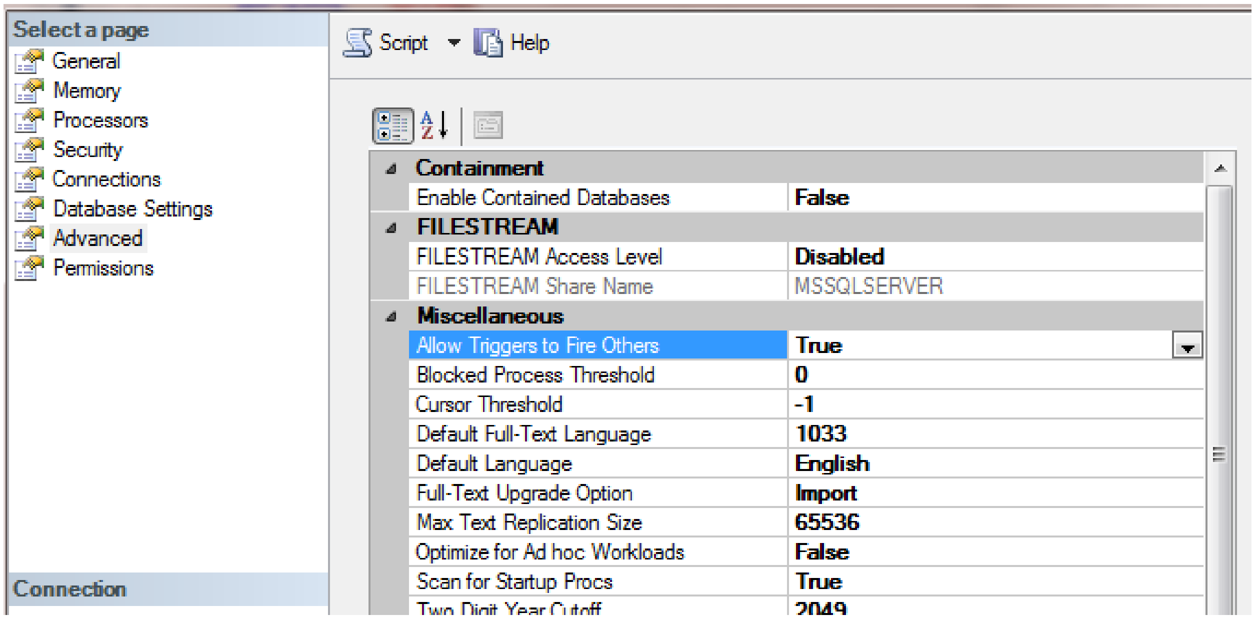
The records modified by the DML statement can be accessed by the trigger code via special logical tables. SQL server provides the following two logical tables:

- **'deleted'**: holds the records deleted form the table in case of a DELETE trigger or the *original* (old) records updated in case of an UPDATE trigger. An update is logically equivalent to a delete followed by an insert. The deleted table is empty in case of an INSERT trigger.
- **'inserted'**: holds the records inserted by an INSERT statement or the *new* records updated by an UPDATE trigger. The inserted table is empty in case of a DELETE trigger.

SQL server also supports the **update**([field name]) function available in INSERT or UPDATE triggers that returns true if the DML statement changed the specified field. The field cannot be a computed column.

If the trigger raises an error, the DML statement is rolled back.

A trigger may run code that invokes other triggers or even the same trigger in a recursive manner, up to 32 levels on MS SQL server. This feature is controlled via the nested triggers server option (see below).



Containment	
Enable Contained Databases	False
FILESTREAM	
FILESTREAM Access Level	Disabled
FILESTREAM Share Name	MSSQLSERVER
Miscellaneous	
Allow Triggers to Fire Others	True
Blocked Process Threshold	0
Cursor Threshold	-1
Default Full-Text Language	1033
Default Language	English
Full-Text Upgrade Option	Import
Max Text Replication Size	65536
Optimize for Ad hoc Workloads	False
Scan for Startup Procs	True
Two Digit Year Cutoff	2049

Cases when the use of a DML trigger is recommended

- Administration functions such as maintaining a log or keeping old values of changed records in backup tables.
- Enforcing data integrity rules that follow from the business logic and that are beyond the scope of simple primary key, foreign key or check constraints. Example in the Northwind database:
  - We do not send heavy packages overseas. Therefore we refuse orders with a freight over 200 kg that has a ShipCountry not equal to USA. This can be implemented by an INSERT AFTER or INSERT INSTEAD OF trigger on the Orders table. Such checks should of course be built into the client software, however, database level integrity enforcement can prevent application errors or hacking.
- Automating business workflow processes. Examples in the Northwind database:
  - We send an automated email to the customer when the shipping date is decided i.e. when the ShippedDate field of an order is set (UPDATE trigger)
  - We automatically send an order to our wholesale supplier when the UnitsInStock of a Product drops below the ReorderLevel (UPDATE or INSERT trigger)
  - We automatically update the UnitsInStock field of the Products table when the quantity field in a corresponding Order Detail record changes (UPDATE trigger)

PRACTICE: write an UPDATE trigger for the Order Details table. When the quantity changes, update the UnitsInStock of the product. You can assume that only one Order Details record is updated at a time.

PRACTICE: Assume that in the problem above more than Order Details records are updated at a time.

WARNING: the operation of triggers is ‘silent’, and severe problems may result from forgetting about them. For example, if the administrator restores the Order Items tables from a backup copy with an UPDATE statement without first disabling the trigger…

For more examples on SQL Server triggers see:

- http://sqlhints.com/2016/02/28/inserted-and-deleted-logical-tables-in-sql-server/

Tight coupling

In the example below we create a new insert trigger on the Orders table that runs long and throws an exception, thus disabling the order saving process.

```
drop trigger tr_demo_bad
go
create trigger tr_demo_bad on orders for insert as
declare @orderid int
select @orderid=OrderID from inserted
print 'New order ID: ' + cast(@orderid as varchar(50))
waitfor delay '00:00:10' --10 s
select 1/0 --we make an error
go
--test #1:  with both last lines commented out
insert  Orders (CustomerID, OrderDate) values ('AROUT', GETDATE())
--restore table
delete Orders where CustomerID='AROUT' and EmployeeID is null
--test #2: recreate the trigger, with the last lines commented out
insert  Orders (CustomerID, OrderDate) values ('AROUT', GETDATE())
--we have long to wait, but there is no error
--restore table
delete Orders where CustomerID='AROUT' and EmployeeID is null
--test #3: recreate the trigger, with all lines
insert  Orders (CustomerID, OrderDate) values ('AROUT', GETDATE())
--we have long to wait, then we have the message:
'New order ID: 11094
Msg 8134, Level 16, State 1, Procedure tr_demo_bad, Line 6 [Batch Start Line 276]
Divide by zero error encountered.
The statement has been terminated.'
select * from Orders where CustomerID='AROUT' and EmployeeID is null
--no such record, because
--the insert statemant has been rolled back -> we crashed the trading system
```

This is exactly what we do NOT want. We implement *loose coupling* instead of *tight coupling*.

The loosely coupled system

The idea is that the trigger only saves the events into a log table. We then process the table with a stored procedure.

The log table and the trigger

The trigger uses the virtual tables *inserted* and *deleted*. This trigger can process multi-record INSERTs and UPDATEs.

```
--the log table
go
--drop table order_log
go
create table order_log (
       event_id int IDENTITY (1, 1) primary key ,
       event_type varchar(50) NOT NULL ,
       order_id int NOT NULL ,
       orderitem_id int NULL ,
       status int NOT NULL default(0),
       time_created datetime NOT NULL default(getdate()) ,
        time_process_begin datetime NULL ,
        time_process_end datetime NULL ,
        process_duration as datediff(second, time_process_begin, time_process_end)
)
go
drop trigger tr_log_order
go
create trigger tr_log_order ON Orders for insert, update as
declare @orderid int
select @orderid=orderid from inserted --there can be more then a single record in inserted
print 'OrderID of the LAST record: ' + cast(@orderid as varchar(50))
if update(orderid) begin --if the orderid has changed, then this is an INSERT
        print 'Warning: new order'
        insert order_log (event_type, order_id)  --status, time_created use default
               select 'new order', orderid from inserted
end else if update(shipaddress) or update(shipcity) begin --shipaddress or shipcity has
changed
        print 'Warning: address changed'
        insert order_log (event_type, order_id)
               select 'address changed', orderid from inserted
end else begin  --other change
        print 'Warning: other change'
        insert order_log (event_type, order_id)
               select 'other change', orderid from inserted
end
go

--test #1
insert  Orders (CustomerID, OrderDate) values ('AROUT', GETDATE())
select * from order_log
--we have one new record in the log table

--test #2
insert  Orders (CustomerID, OrderDate) values ('AROUT', GETDATE()), ('HANAR', GETDATE())
select * from order_log
--we have two new records in the log table

--test #3
update Orders set ShipVia = 3 where OrderID in (11097, 11096) --these are the IDs of test #2
select * from order_log
--we have two new records of the type 'other change'

--restore the tables
delete Orders where CustomerID in ('AROUT', 'HANAR') and EmployeeID is null
delete order_log
```

The stored procedure for processing new orders

We expect that the items of a new order are inserted subsequently after the order record is created.

```
--a simple stored procedure that processes a new order
--and returns 0 if all of its items could be committed to the inventory without error
--demonstrating also the use of output parameters
drop proc  sp_commit_new_order_to_inventory
go
create procedure sp_commit_new_order_to_inventory
@orderid int,
@result int output
as
begin try
        update products set unitsInStock = unitsInStock - od.quantity
        from products p inner join [Order Details] od on od.ProductID=p.ProductID
        where od.OrderID=@orderid
        set @result=0
end try
begin catch
        print '  Inventory error: '+ ERROR_MESSAGE() + ' (' + cast(ERROR_NUMBER() as
varchar(20)) + ')'
        set @result=1
end catch
go

--test
select * from order_log --11097
select * from Products where ProductID=10 --unitsinstock =31
select * from Products where ProductID=9 --unitsinstock =29
insert [Order Details]  (orderid, productid, quantity, UnitPrice, Discount)
values (11097, 9, 10, 30, 0),(11097, 10, 40, 30, 0)  --the second item will cause an error in
sp_commit_new_order_to_inventory
go
declare @res int
exec sp_commit_new_order_to_inventory 11097, @res output
print @res
exec sp_commit_new_order_to_inventory 11096, @res output
print @res
go
--check: no change in unitsinstock (OK)
select * from Products where ProductID=10 --unitsinstock =31
select * from Products where ProductID=9 --unitsinstock =29
```

The stored procedure for processing the event log

Since completely different actions are to be taken depending on the event type, we use a cursor to iterate the order log.

```
--stored procedure for processing the order_log
--drop proc sp_order_process
go
create proc sp_order_process as
declare @event_id int, @event_type varchar(50), @order_id int, @result int
declare cursor_events cursor forward_only static
        for
        select  event_id, event_type, order_id
        from order_log where status=0 --we only care for the unprocessed events

set xact_abort on
set nocount on
open cursor_events
fetch next from cursor_events into @event_id, @event_type, @order_id
while @@fetch_status = 0
begin
        print 'Processing event ID=' + cast(@event_id as varchar(10)) + ', Order ID=' +
cast(@order_id as varchar(10))
        update order_log set time_process_begin=getdate() where event_id=@event_id
        begin tran
        set @result = null
        if @event_type = 'new order' begin
                print '  Processing new order...'
                exec sp_commit_new_order_to_inventory @order_id, @result output
        end else if @event_type = 'address changed' begin
                print '  Processing address changed...'
                waitfor delay '00:00:01' --we only simulate the processing of other event types
                set @result=0
        end else if @event_type = 'other change' begin
                print '  Processing other change...'
                waitfor delay '00:00:01'
                set @result=0
        end else begin
                print '  Unknown event type...'
                waitfor delay '00:00:01'
                set @result=1
        end

        if @result=0 begin
                print 'Event processing OK'
                commit tran
        end else begin
                print 'Event processing failed'
                rollback tran
        end
        print ''
        update order_log set time_process_end=getdate(),
                status=case when @result=0 then 2 else 1 end
                where event_id=@event_id
        fetch next from cursor_events into @event_id, @event_type, @order_id
end
close cursor_events deallocate cursor_events
go

--teszt
update order_log set status=0
select *from orders where EmployeeID is null
select * from order_log
exec dbo.sp_order_process
select * from order_log

--we get:
Processing event ID=5, Order ID=11097
  Processing new order...
  Inventory error: The UPDATE statement conflicted with the CHECK constraint etc.
Event processing failed

Processing event ID=6, Order ID=11096
  Processing new order...
Event processing OK
```

The scheduled job that calls the event log processor

We implement the job using the SSMS GUI and check its operation in the Job Activity Monitor.

PRACTICE: create a loosely coupled solution that monitors the Products table and orders new supply from the associated Supplier when the UnitsinStock value falls below the value specified in the ReorderLevel field.

3. Replication, log shipping and failover

In the loose coupling case study we were in fact implementing a special form of *replication*.

Replication concepts and architecture

Replica means a copy of the original. In database technology, replication is used to automate the copying and merging of data from or to multiple sources. The components of the replication metaphor are as follows.

- The **publisher** is the entity (a database server) that has data to be shared. Such data is organized into **publications**. Each publication contains one or more **articles**. The articles can be tables or parts of tables, stored procedures or other database objects.
- The **subscriber** is the entity that subscribes to publications. It can be the same database server as the publisher or another server. Several subscribers, possibly on different servers, may subscribe to the same publication.
- The **subscription** may have various modalities with respect to the way and scheduling of copying. It can also include data filters or on-the-fly data transform steps. It can be a **push** or a **pull** subscription. The pull subscriptions are created at, and scheduled by, the subscriber.

The main types and application scenarios of replication are as follows.

- **Snapshot** replication. After an initial snapshot of the articles, the copied objects will be dropped and re-created on the subscriber each time the data is refreshed, regardless of whether there was any change in the publication. Applicable for copying parts of an OLTP database off to a data warehouse or a reporting server, scheduled out of office hours. An example is sending data generated during the day overnight ('point in time reports'). Since several subscriptions may point to the same destination database, replication can be used as an ETL (extract-transform-load) mechanism if SQL Server technology is used by all publishers. WARNING: due to the drop/re-create mechanism, objects at the subscriber may be temporarily inaccessible. The latency of the data must also be tolerated. *All other replication types below are initialized with a snapshot*.
- **Transactional** replication. It copies only the data that has been changed, and it can be configured for near real time data synchronization. A primary key on the replicated tables is needed. Applicable when considerable latency is a problem and when we do not want to move unchanged data. An example is the off-site branches of a company that have their own local servers holding only parts of the central database relevant for their operation. Such an architecture improves site autonomy and robustness of the information system.
- **Merge** replication. In this scheme the subscribers may themselves generate changes to the data and there is a mechanism in place that distributes these changes to all parties and merges them into a consistent database. The merging process may also involve conflict resolution. In order to identify records across multiple servers, the tables must have a field of UNIQUEIDENTIFIER data type with ROWGUIDCOL[3] property. A typical scenario is the case of traveling businesspeople who are not always connected the central database. The changes they make on their local database is merged with others' changes automatically.

The type of the replication is always determined by the publication.

[3] Works like an identity column without a seed and with globally unique values.

Replication technology is based on scheduled jobs and, in the case of merge replication, triggers on published articles.

Replication is **not recommended** when an exact copy of a whole database is to be maintained on a remote server and the goal is to improve availability and reliability, because log shipping and the Always-On technology of SQL server 2012 and later offer a simpler and more robust solution.

WARNING: though replication prepares multiple copies of the data, it is not a replacement for backups and disaster recovery planning.

There are three **server roles** in replication, the *publisher*, the *distributor* and the *subscriber*. All three roles may be taken by the same server instance when a local database is replicated into another local database, or by different instances. In more realistic setups, the distributor role is taken by another server to offload the publisher. The **distributor** is responsible for storing the changed data in a shared folder or a distribution database and for forwarding the data to the subscribers. A publisher can have only one distributor, but a distributor may serve several subscribers.

In **bi-directional** or **updatable** replication the subscriber may be allowed to make changes on the publisher as well.

SQL server implements replication functionality with various **agents**. These agents are jobs running under the supervision of SQL Server Agent.

- **The Snapshot agent** generates the snapshot and stores it in the **snapshot folder** at the distributor. The agent uses the bcp (bulk copy) utility to copy the articles of the publication.
- **The Distribution agent**. In snapshot replication this agent applies the snapshot to the subscriber and in transactional replication it runs the transactions held in the **distribution database** on the subscriber. The distribution database is a system database on the distributor, therefore you can find it in the System Databases group. This agent runs at the subscriber for pull subscriptions and it runs at the publisher for push subscriptions.
- **The Log reader agent** reads the transaction log at the publisher and copies the relevant transactions from the log to the distribution database. It is used only in transactional replication. There is a separate agent for each database published.
- **The Queue reader agent** copies changes made by the subscribers to the publisher in an *updatable* or *bi-directional* transactional replication.
- **The Merge agent** merges incremental changes that occur at both the subscriber and the publisher in merge replication. Detecting changes is based on triggers. The merge agent is not used in transactional replication.

Except for a pull subscription, all agents run at the distributor.

Snapshot replication

First of all, configure the test environment. For the replication examples to work as expected, you need three ‘named’ MS SQL Server instances installed on the same server machine. They should be named PRIM, SECOND and THIRD.

Scenario: we want to replicate the orders of the American customers to another database on the same server (Principal) to refresh the reporting data warehouse overnight. We choose snapshot replication.

Creating the publication

1. Connect to the Principal server and create a new database called Northwind if it does not exist. Run the create dump of the Northwind database.

2. In order to avoid the error "*Cannot execute as the database principal because the principal "dbo" does not exist,* …" etc. that is due to an incomplete restore of the Northwind database, execute the following script on the Northwind database:

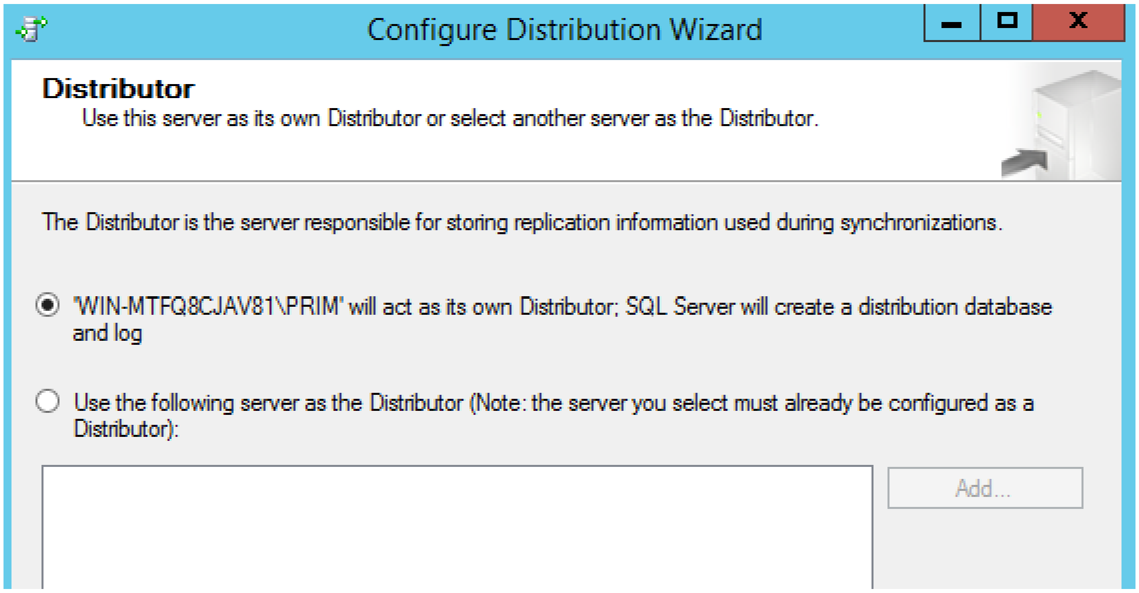
   ```
   EXEC sp_changedbowner 'sa';
   ALTER AUTHORIZATION ON DATABASE::northwind TO sa;
   ```

3. Create another empty database called nw_repl, also on the PRIM server.

4. Start SQL Server Agent if it is not running

5. Selecting Replication -> Configure Distribution, set the PRIM to be its own distributor. Note that the snapshot folder will be this:

   C:\Program Files\Microsoft SQL Server\MSSQL14.PRIM\MSSQL\ReplData



Configure Distribution Wizard

Distributor

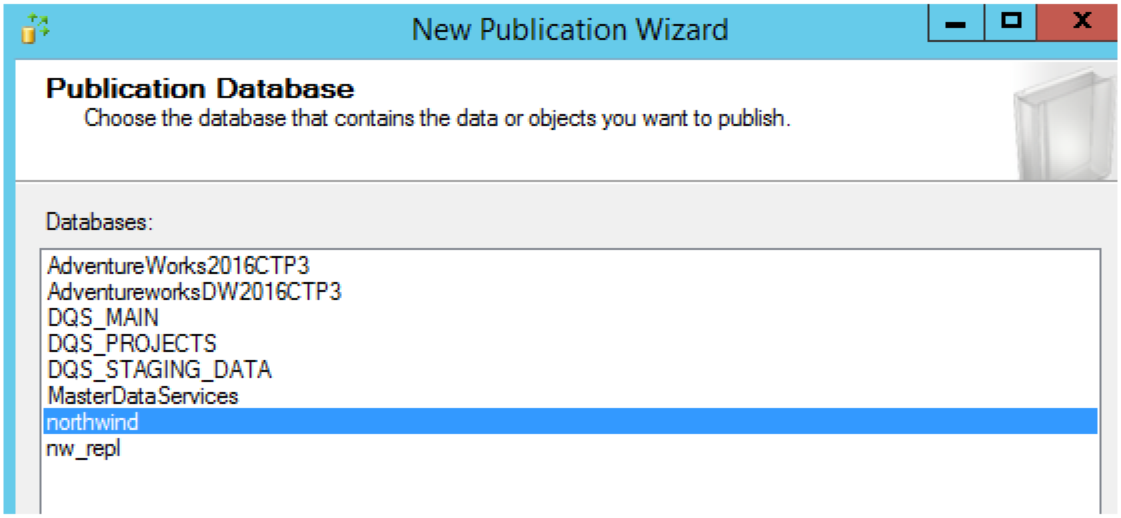
Use this server as its own Distributor or select another server as the Distributor.

The Distributor is the server responsible for storing replication information used during synchronizations.

● 'WIN-MTFQ8CJAV81\PRIM' will act as its own Distributor; SQL Server will create a distribution database and log

○ Use the following server as the Distributor (Note: the server you select must already be configured as a Distributor):

6. Start the New publication wizard and select northwind as the publication database:



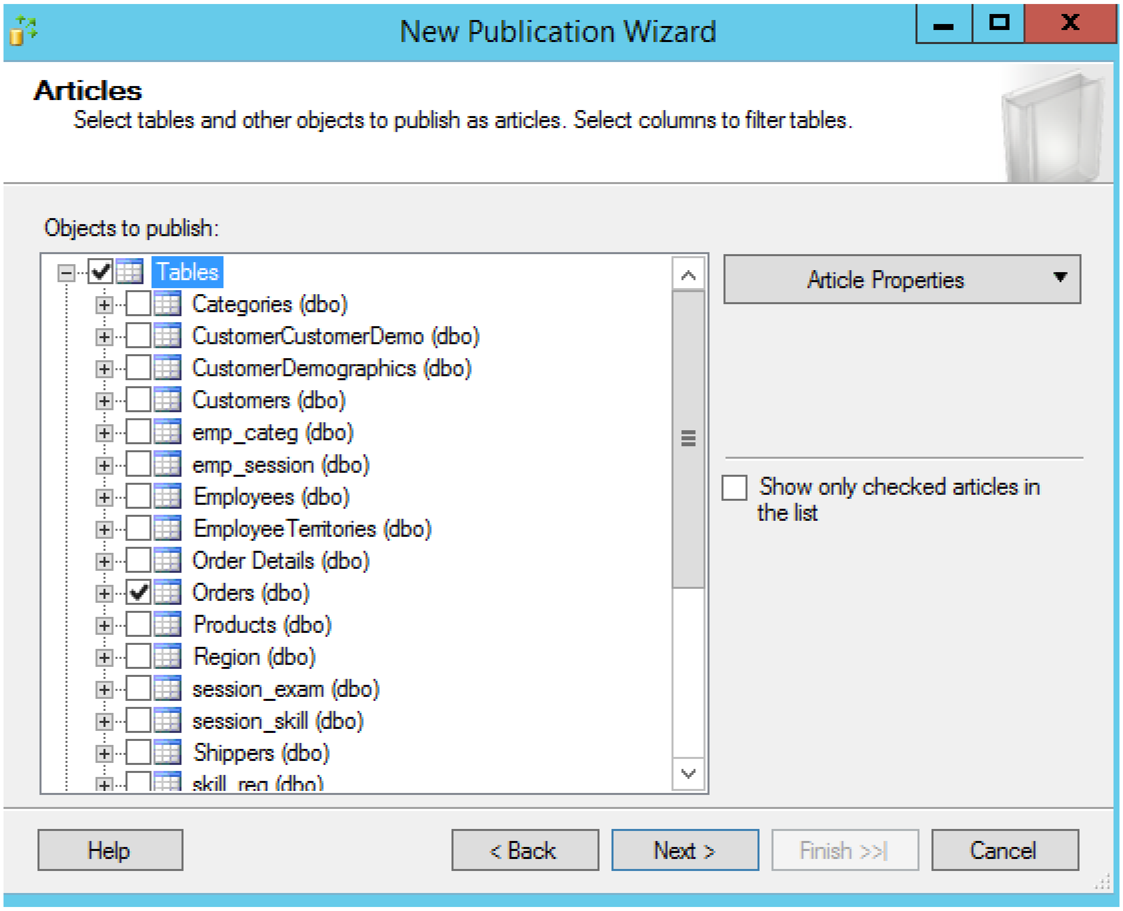
Publication Database

Choose the database that contains the data or objects you want to publish.

Databases:

AdventureWorks2016CTP3
AdventureworksDW2016CTP3
DQS_MAIN
DQS_PROJECTS
DQS_STAGING_DATA
MasterDataServices
northwind
nw_repl

7. On the next panel, select Snapshot publication, then select the Orders table as the single article of the publication:



Articles

Select tables and other objects to publish as articles. Select columns to filter tables.

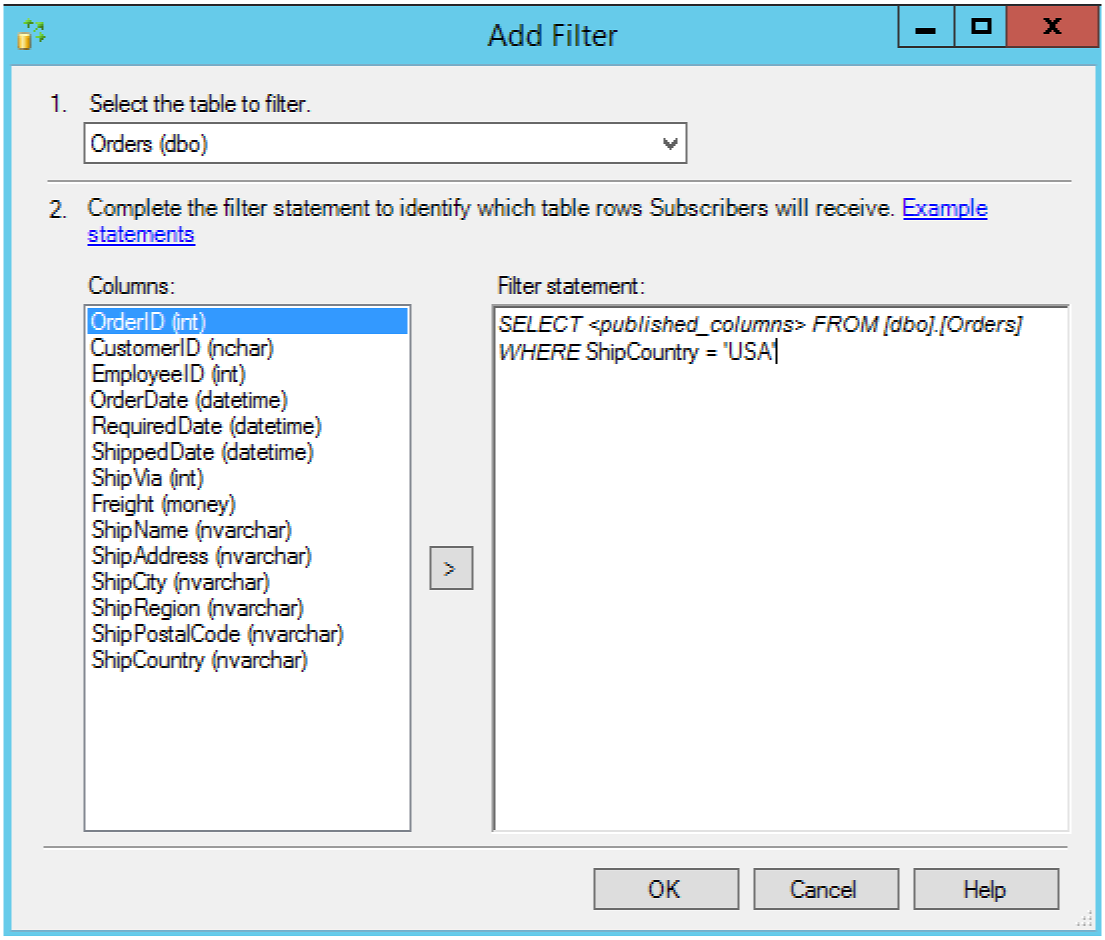
Objects to publish:

- ☑ Tables
  - □ Categories (dbo)
  - □ CustomerCustomerDemo (dbo)
  - □ CustomerDemographics (dbo)
  - □ Customers (dbo)
  - □ emp_categ (dbo)
  - □ emp_session (dbo)
  - □ Employees (dbo)
  - □ EmployeeTerritories (dbo)
  - □ Order Details (dbo)
  - ☑ Orders (dbo)
  - □ Products (dbo)
  - □ Region (dbo)
  - □ session_exam (dbo)
  - □ session_skill (dbo)
  - □ Shippers (dbo)
  - □ skill_req (dbo)

□ Show only checked articles in the list

8. On the Filter table dialog, choose Add

9. Complete the filter statement to filter out American customers



1. Select the table to filter.

Orders (dbo)

2. Complete the filter statement to identify which table rows Subscribers will receive. Example statements

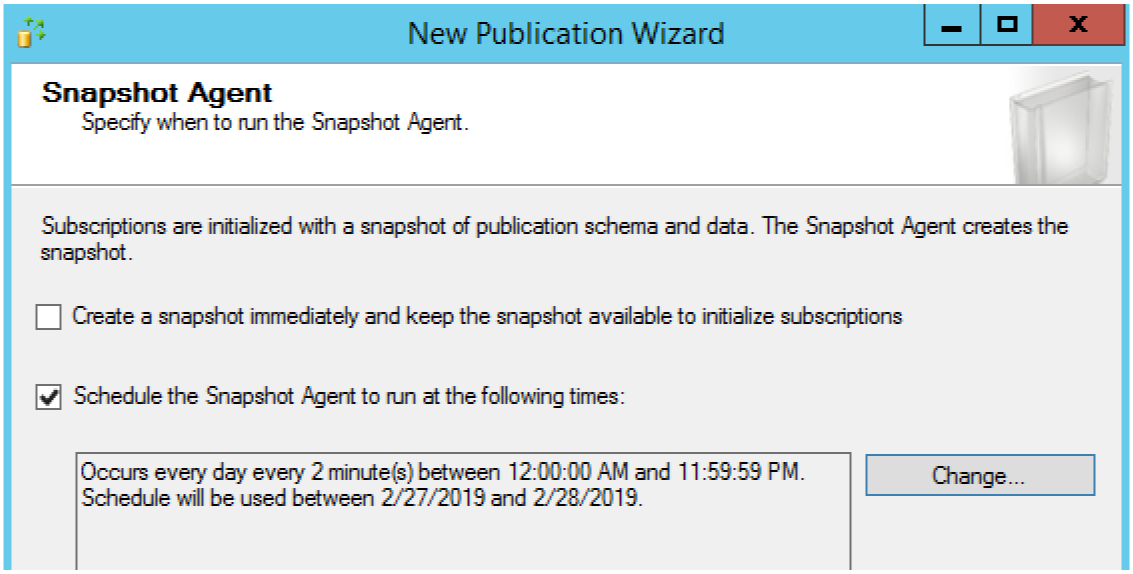
Columns:

- OrderID (int)
- CustomerID (nchar)
- EmployeeID (int)
- OrderDate (datetime)
- RequiredDate (datetime)
- ShippedDate (datetime)
- ShipVia (int)
- Freight (money)
- ShipName (nvarchar)
- ShipAddress (nvarchar)
- ShipCity (nvarchar)
- ShipRegion (nvarchar)
- ShipPostalCode (nvarchar)
- ShipCountry (nvarchar)

Filter statement:

```
SELECT <published_columns> FROM [dbo].[Orders]
WHERE ShipCountry = 'USA'
```

10. Specify that the snapshot agent should run every two minutes by selecting Change on the next panel. Note: *We use this short time interval **only for demo** purposes. The agent runs the* bcp *utility which places a lock on the whole table until it finishes the copying in order to guarantee data consistency. This means the blocking of all other transactions that may wish to modify the table. Snapshot generation in production systems should be scheduled considering performance implications.*



New Publication Wizard

Snapshot Agent

Specify when to run the Snapshot Agent.

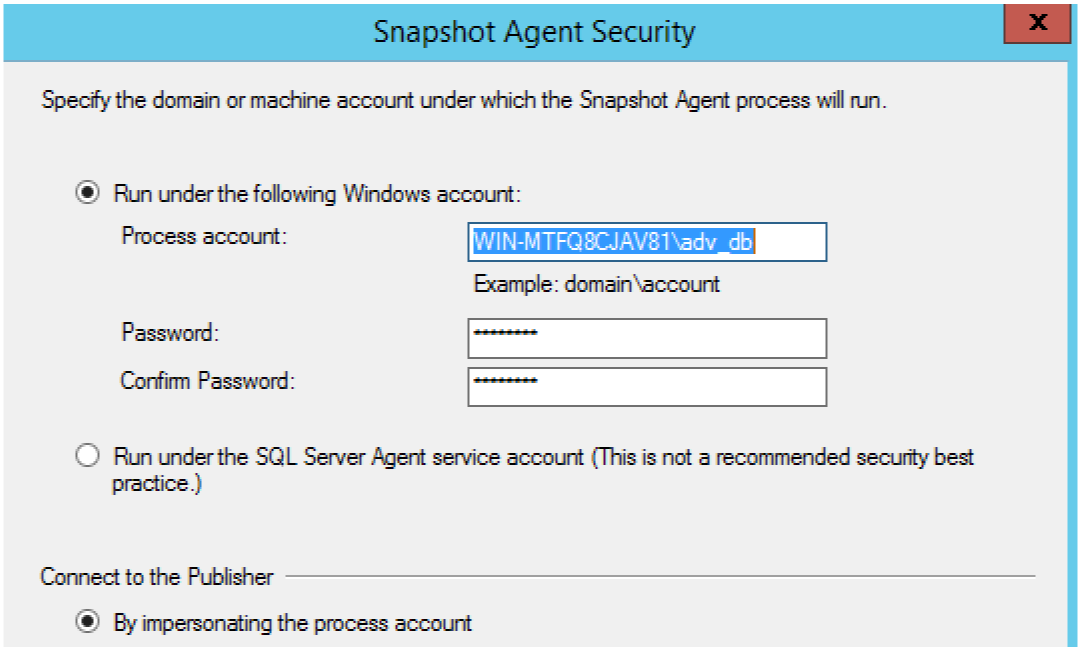
Subscriptions are initialized with a snapshot of publication schema and data. The Snapshot Agent creates the snapshot.

☐ Create a snapshot immediately and keep the snapshot available to initialize subscriptions

☑ Schedule the Snapshot Agent to run at the following times:

Occurs every day every 2 minute(s) between 12:00:00 AM and 11:59:59 PM. Schedule will be used between 2/27/2019 and 2/28/2019.

11. We now have to specify agent security. On the Security settings tab, set your own user credentials and impersonate process account. This is the simplest way of ensuring that the snapshot agent will have write permission to the snapshot folder. **Note**: you might wonder why the SQL Server Agent service or the SQLSERVER service uses a low privilege non-administrator Windows account. The reason for this is that in this way an attacker who has successfully cracked the DBMS has less chance to corrupt the whole server.



Snapshot Agent Security

Specify the domain or machine account under which the Snapshot Agent process will run.

● Run under the following Windows account:

Process account: WIN-MTFQ8CJAV81\adv_db

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

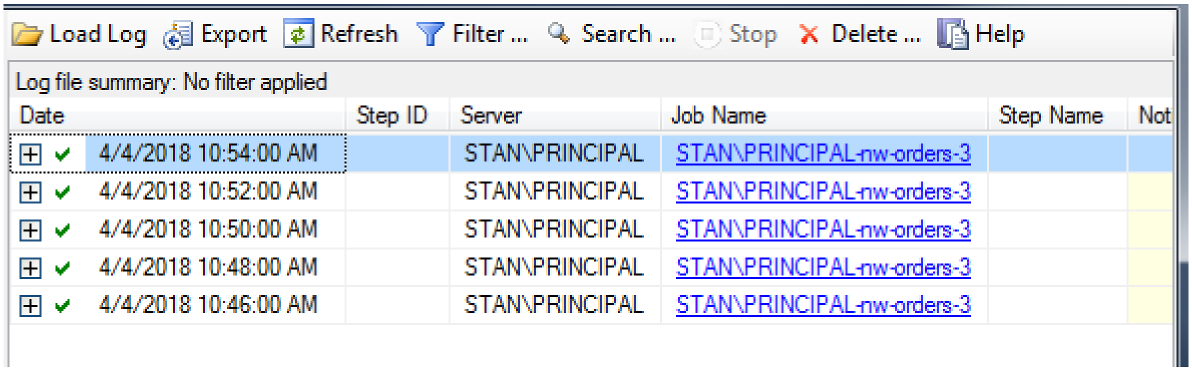
● By impersonating the process account

12. On the next panel, select create the publication and name it ‘orders’.

Checking the publication

The new publication appears under Local publications. The snapshot folders are created at C:\Program Files\Microsoft SQL Server\MSSQL14.PRIM\MSSQL\repldata\unc\WIN-MTFQ8CJAV81$PRIM_NORTHWIND_TEST, but no actual snapshot was not generated because no subscriptions needed initialization yet.

Check the new job that appears under SQL Server Agent jobs. The job history shows that the agent is run periodically as configured.

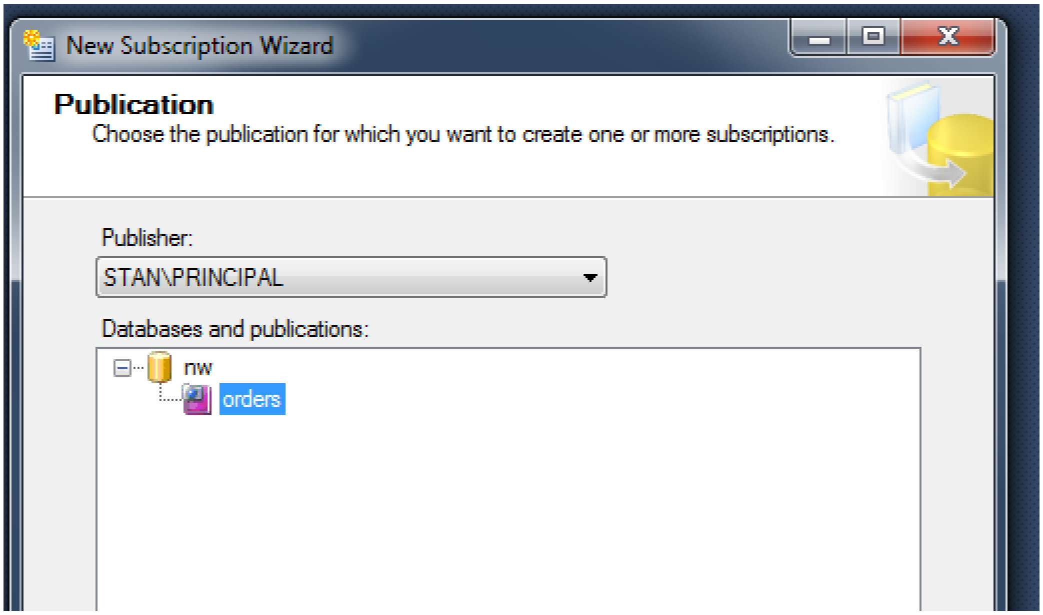


Log file summary: No filter applied

	Date	Step ID	Server	Job Name	Step Name	Not
✓	4/4/2018 10:54:00 AM		STAN\PRINCIPAL	STAN\PRINCIPAL-nw-orders-3		
✓	4/4/2018 10:52:00 AM		STAN\PRINCIPAL	STAN\PRINCIPAL-nw-orders-3		
✓	4/4/2018 10:50:00 AM		STAN\PRINCIPAL	STAN\PRINCIPAL-nw-orders-3		
✓	4/4/2018 10:48:00 AM		STAN\PRINCIPAL	STAN\PRINCIPAL-nw-orders-3		
✓	4/4/2018 10:46:00 AM		STAN\PRINCIPAL	STAN\PRINCIPAL-nw-orders-3		

Creating a push subscription

1. Start the new subscription wizard from the pop-up menu on the orders publication. Select the orders publication as the source



Publication

Choose the publication for which you want to create one or more subscriptions.

Publisher:

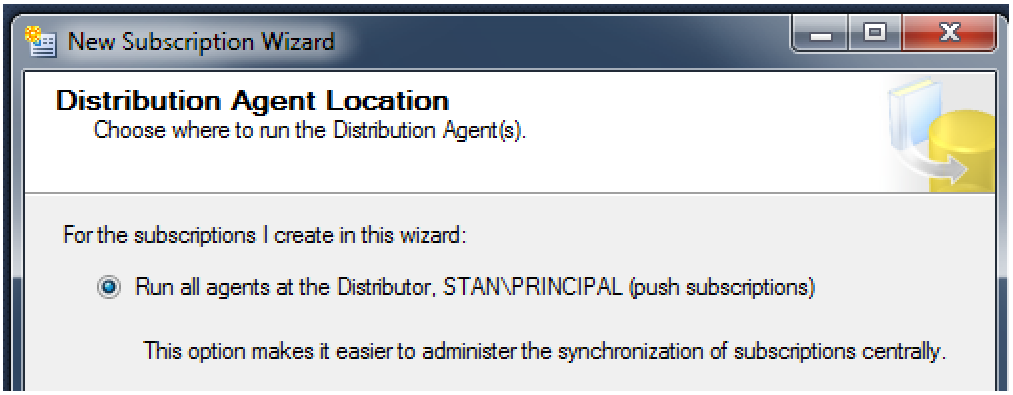
STAN\PRINCIPAL

Databases and publications:

nw

orders

2. On the next panel, choose Run all agents at the distributor for push subscription



Distribution Agent Location

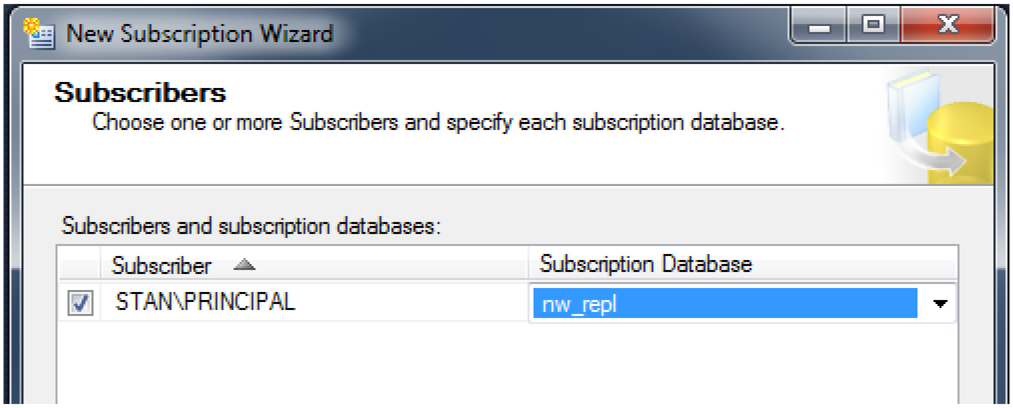
Choose where to run the Distribution Agent(s).

For the subscriptions I create in this wizard:

● Run all agents at the Distributor, STAN\PRINCIPAL (push subscriptions)

This option makes it easier to administer the synchronization of subscriptions centrally.

3. Specify the same server as the subscriber and the nw_repl as the subscription database



New Subscription Wizard

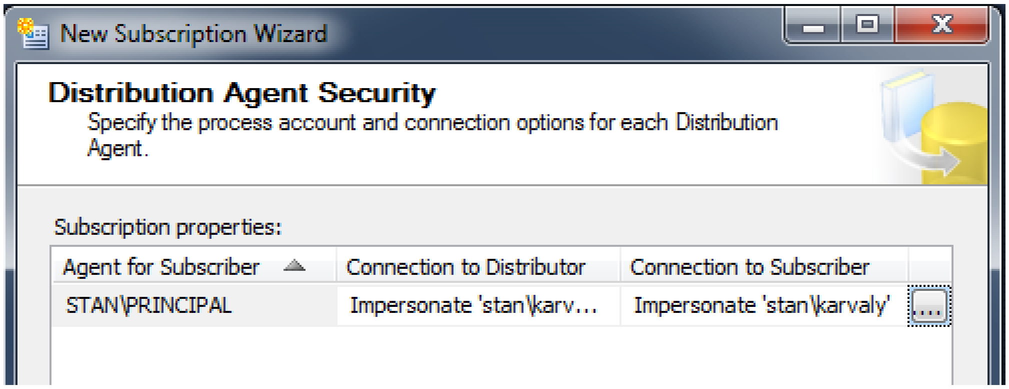
Subscribers

Choose one or more Subscribers and specify each subscription database.

Subscribers and subscription databases:

	Subscriber	Subscription Database
☑	STAN\PRINCIPAL	nw_repl

4. Set the security of the distribution agent the same way as the snapshot agent



New Subscription Wizard

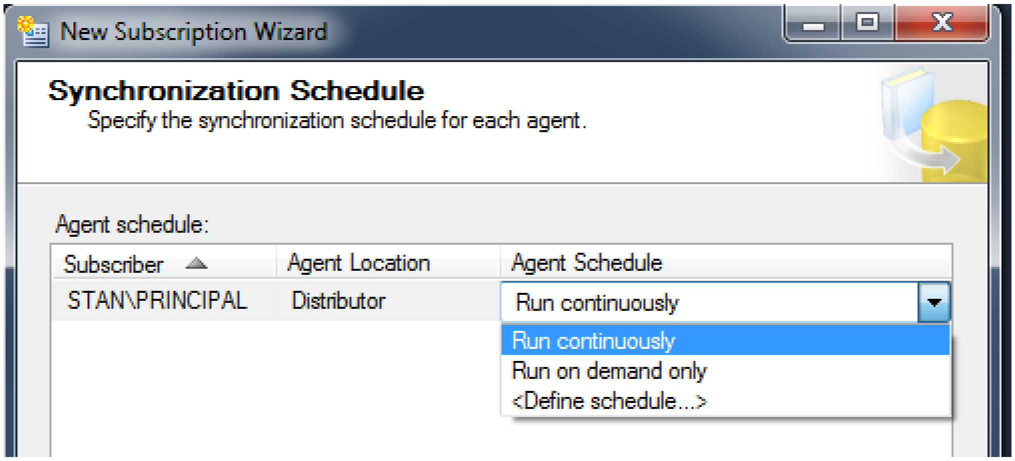
Distribution Agent Security

Specify the process account and connection options for each Distribution Agent.

Subscription properties:

Agent for Subscriber	Connection to Distributor	Connection to Subscriber
STAN\PRINCIPAL	Impersonate 'stan\karv...	Impersonate 'stan\karvaly'

5. For the schedule, select Run continuously to provide minimal latency



New Subscription Wizard

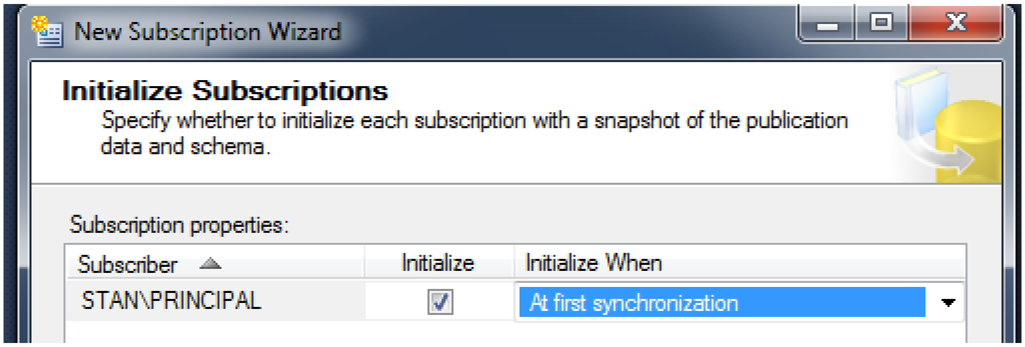
Synchronization Schedule

Specify the synchronization schedule for each agent.

Agent schedule:

Subscriber	Agent Location	Agent Schedule
STAN\PRINCIPAL	Distributor	Run continuously

6. On the next panel, select initialization. This will generate the first snapshot in the distribution folder.



New Subscription Wizard

Initialize Subscriptions

Specify whether to initialize each subscription with a snapshot of the publication data and schema.

Subscription properties:

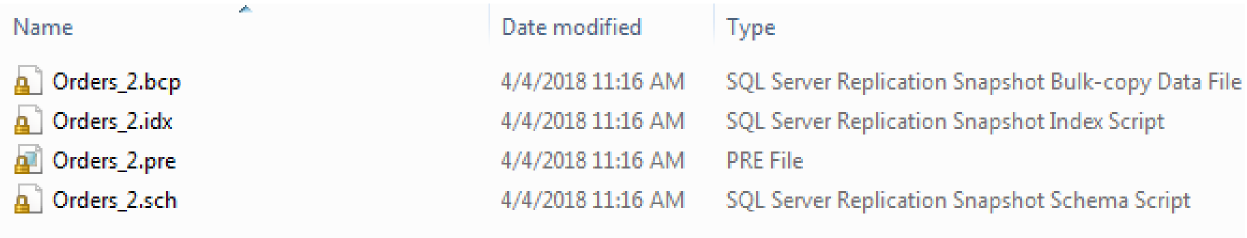
Subscriber	Initialize	Initialize When
STAN\PRINCIPAL	☑	At first synchronization

7. Finish creating the new subscription

Note: you can change the properties of subscriptions and publications later if you select Properties from their pop-up menu.

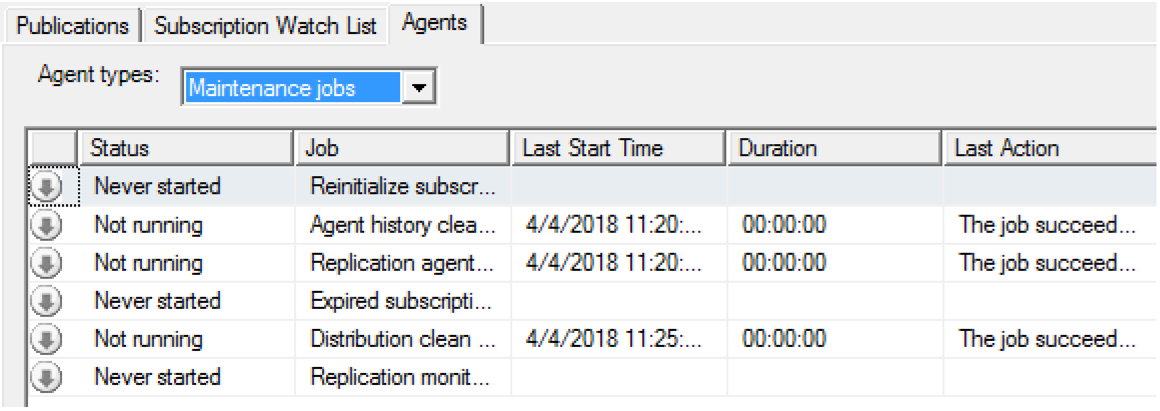
Checking the subscription

1. Locate the new Orders table in the nw_repl database and check that it contains the USA orders
2. Check the contents of the snapshot folder. **Bulk copy** is a fast method SQL server uses to insert data directly into database files.



Name	Date modified	Type
Orders_2.bcp	4/4/2018 11:16 AM	SQL Server Replication Snapshot Bulk-copy Data File
Orders_2.idx	4/4/2018 11:16 AM	SQL Server Replication Snapshot Index Script
Orders_2.pre	4/4/2018 11:16 AM	PRE File
Orders_2.sch	4/4/2018 11:16 AM	SQL Server Replication Snapshot Schema Script

3. Start the Replication monitor from the pop-up menu of the new subscription and check the publication and the subscription status. You can also review the active replication agents here:



Agent types: Maintenance jobs

	Status	Job	Last Start Time	Duration	Last Action
	Never started	Reinitialize subscr...			
	Not running	Agent history clea...	4/4/2018 11:20:...	00:00:00	The job succeed...
	Not running	Replication agent...	4/4/2018 11:20:...	00:00:00	The job succeed...
	Never started	Expired subscripti...			
	Not running	Distribution clean ...	4/4/2018 11:25:...	00:00:00	The job succeed...
	Never started	Replication monit...			

8. Open the Job activity monitor. The distribution agent appears as a new job in the list, with a status of Executing all the time
9. Change the first USA record of the orders table at the publisher with an UPDATE statement. The change appears in the replicated table shortly (ca. 30 seconds) after the snapshot agent is run the next time.
10. Finally, delete the subscription and the publication. This can be accomplished by selecting Generate scripts form the popup menu of the Replication group, specifying 'To drop…' and running the script in an editor. Alternatively, you can delete the objects one-by-one manually.
11. The replicated tables at the subscriber will not be deleted by deleting the subscription, so delete the Orders table manually from the nw_repl database.

PRACTICE: create a push snapshot publication into the nw_repl table that copies those employees from the Employees table whose title is Sales representative. Verify the correct operation, then delete all related objects.

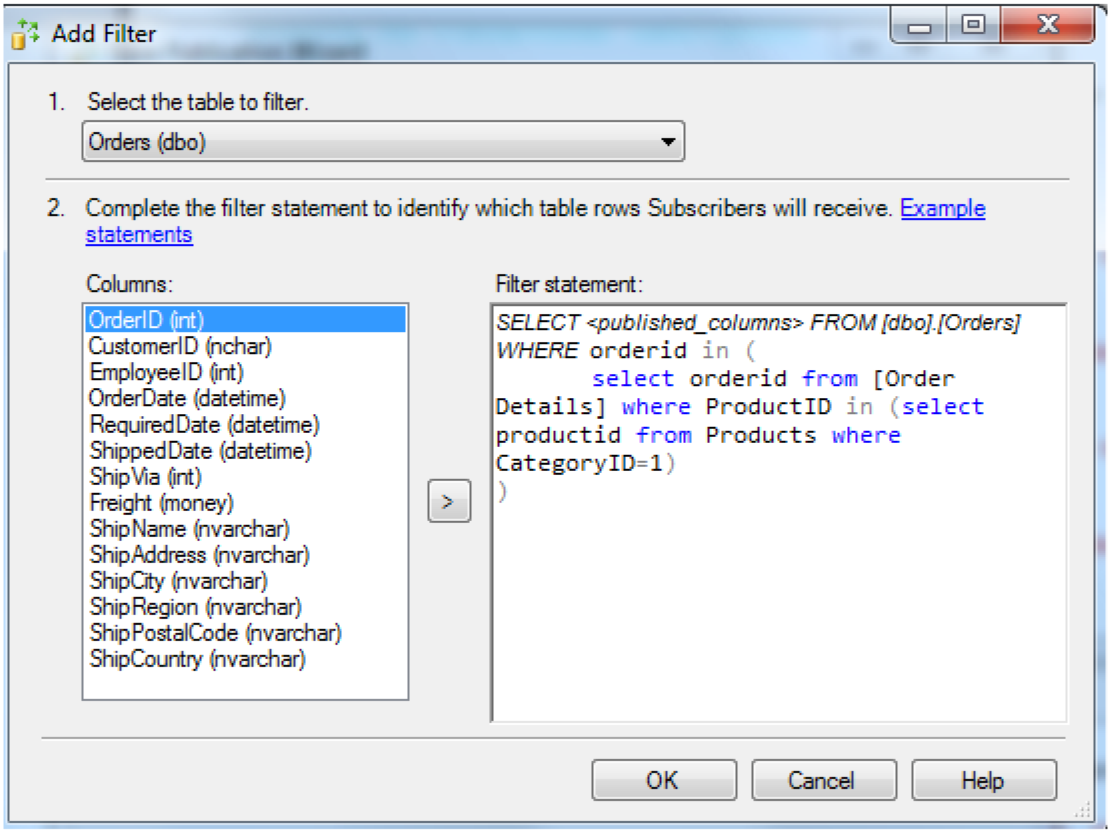
Transactional replication

Scenario: we wish to create a near-real time (scheduled) loose coupling between the central Northwind database and an off-site division that deals only with the products of the category 'Beverages' (CategoryID=1). We replicate only orders and order items that are beverages via transactional replication.

First we implement this demo on a single server (Principal). The filter condition above can be defined as follows:

```
select * from [Order Details] where ProductID in (select productid from Products where
CategoryID=1)

select * from orders where orderid in (
        select orderid from [Order Details] where ProductID in (select productid from Products
where CategoryID=1)
)
```

1. You can reset the replication configuration by selecting Disable publishing and distribution from the Replication menu
2. Define the type of the publication as transactional on the Publication type dialog panel
3. Select the Orders and Order Details tables on the Articles panel
4. On the Filter table panel, add the filter to the two tables one by one by copying the WHERE part from the above queries. For the Orders table:



1. Select the table to filter.

Orders (dbo)

2. Complete the filter statement to identify which table rows Subscribers will receive. Example statements

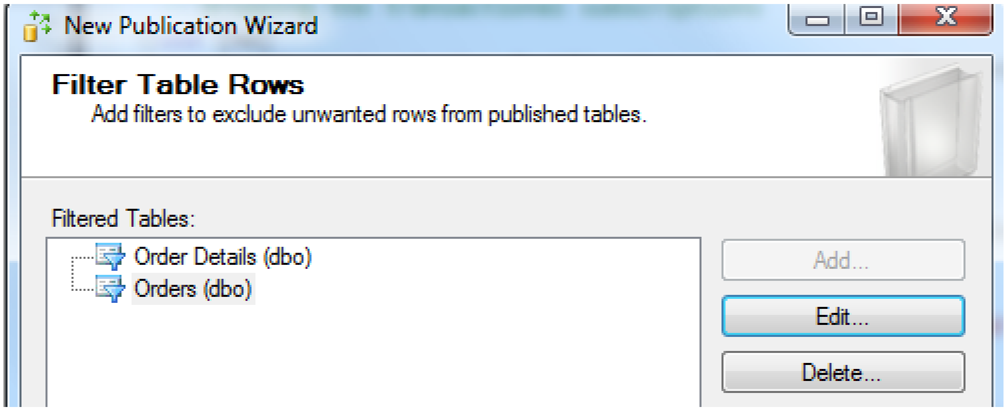
Columns:

OrderID (int)
CustomerID (nchar)
EmployeeID (int)
OrderDate (datetime)
RequiredDate (datetime)
ShippedDate (datetime)
ShipVia (int)
Freight (money)
ShipName (nvarchar)
ShipAddress (nvarchar)
ShipCity (nvarchar)
ShipRegion (nvarchar)
ShipPostalCode (nvarchar)
ShipCountry (nvarchar)

Filter statement:

```
SELECT <published_columns> FROM [dbo].[Orders]
WHERE orderid in (
        select orderid from [Order
Details] where ProductID in (select
productid from Products where
CategoryID=1)
)
```

5. You should have the filters defined for both tables:



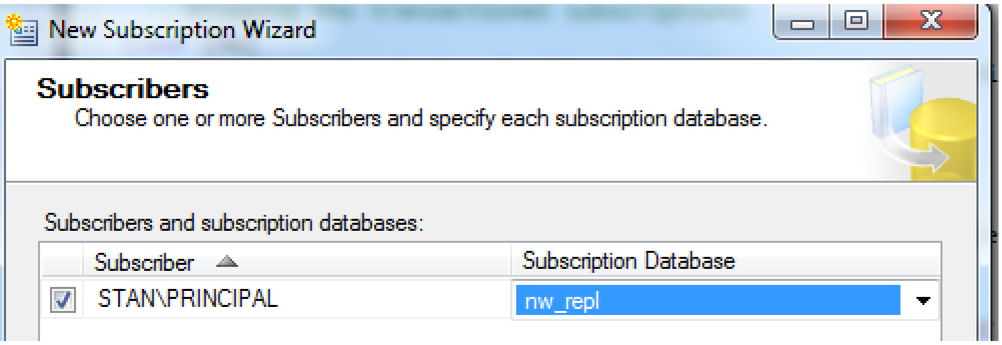
Filter Table Rows

Add filters to exclude unwanted rows from published tables.

Filtered Tables:

- Order Details (dbo)
- Orders (dbo)

6. On the next panel, select Create a snapshot immediately
7. On the panels that follow set the security of the agents the same way as in the previous demo
8. Name the publication nw_trans and finish creating the publication
9. Select New subscription in the pop-up menu of the publication
10. Select the Run all agents at the Distributor on the next panel (push subscription)
11. Select the nw_repl database as the subscription database:



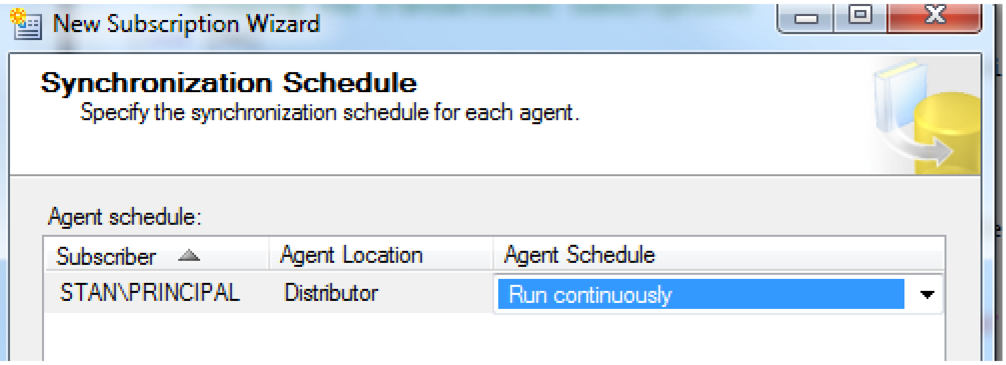
Subscribers

Choose one or more Subscribers and specify each subscription database.

Subscribers and subscription databases:

	Subscriber	Subscription Database
☑	STAN\PRINCIPAL	nw_repl

12. Set the security of the distribution agent the same way as in the previous demo
13. Specify Run continuously for the schedule of the Log reader and Distribution agents:



Synchronization Schedule

Specify the synchronization schedule for each agent.

Agent schedule:

Subscriber	Agent Location	Agent Schedule
STAN\PRINCIPAL	Distributor	Run continuously

14. On the next panel titled Initialize subscriptions, choose Immediately

15. Test the correct operation of the transactional replication. Update the employeeID in the first record of the Orders table in the northwind database and then select the same record in the nw_repl database. You should see the changed value within 10 seconds.

16. Check the operation of the replication agents

17. Clean up the replication by deleting all replication objects

Replication between separate servers[4]

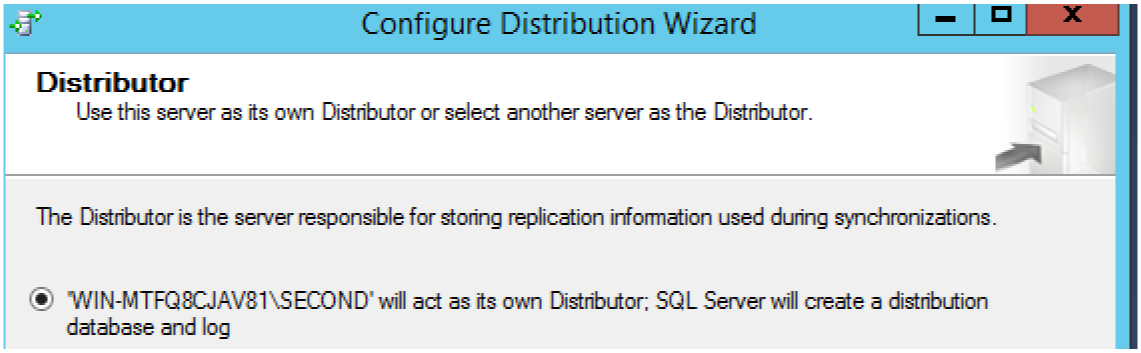
In the next demo we implement the same transactional replication in a more realistic scenario using the PRIM as the publisher, the SECOND server as the distributor and the THIRD as the subscriber, respectively. In an even more realistic scenario, they would be not only separate server instances, but they would also reside on separate server machines. We cannot, however, implement such a scenario in the lab.

Configuring the distributor

1. Start the SECOND and THIRD instances and the SQL Server Agent on SECOND

2. Reset the replication configuration on PRIM by selecting Disable publishing and distribution from the Replication menu

3. In the pop-up menu of Replication on the SECOND instance select Configure Distribution, and accept the first choice. This will create the distribution database on SECOND.


Configure Distribution Wizard

Distributor

Use this server as its own Distributor or select another server as the Distributor.

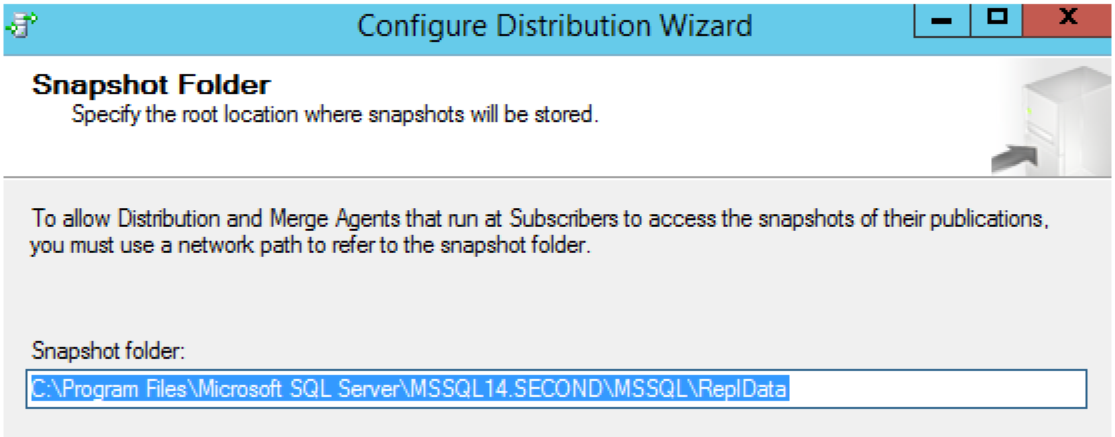
The Distributor is the server responsible for storing replication information used during synchronizations.

● 'WIN-MTFQ8CJAV81\SECOND' will act as its own Distributor; SQL Server will create a distribution database and log

[4] Due to a bug in SQL Server 2019 on Windows 10 systems, the wizard using sp_adddistributor for a remote Distributor uses an empty password instead of the specified one and returns the exception 21768 („The password specified for the @password parameter must be the same when the procedure is executed at the Publisher and at the Distributor”). For this demo to run, execute the sp_adddistributor manually, specifying the right password, like this:

```
use master
exec sp_adddistributor @distributor = 'DESKTOP-....\SECOND', @password = 'type password here'
```

4. On the next panel select that we start the Server Agent manually
5. On the next panel accept the location of the snapshot folder.



Configure Distribution Wizard

Snapshot Folder

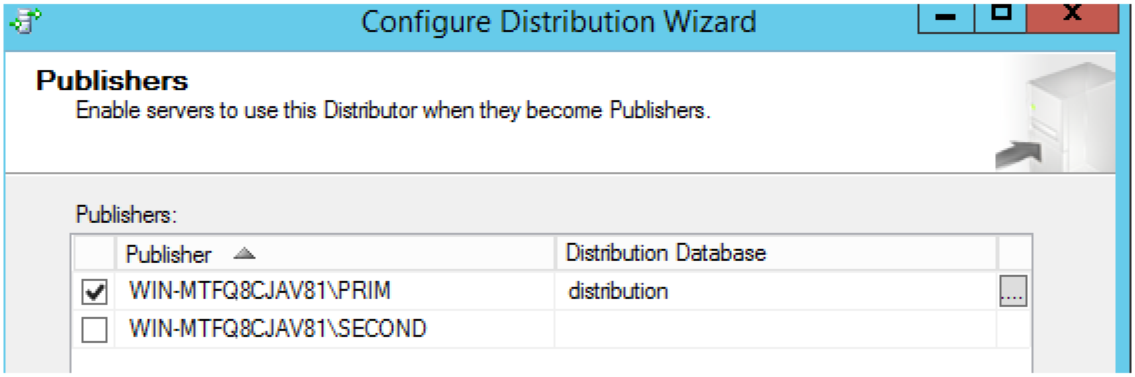
Specify the root location where snapshots will be stored.

To allow Distribution and Merge Agents that run at Subscribers to access the snapshots of their publications, you must use a network path to refer to the snapshot folder.

Snapshot folder:

C:\Program Files\Microsoft SQL Server\MSSQL14.SECOND\MSSQL\ReplData

6. On the next panel accept the defaults for the location and name of the distribution database.
7. We then have to specify which publisher servers are allowed to use this distribution database. On the next pane deselect SECOND (since SECOND will not be publishing) and by pressing Add, add the PRIM instance:



Configure Distribution Wizard

Publishers

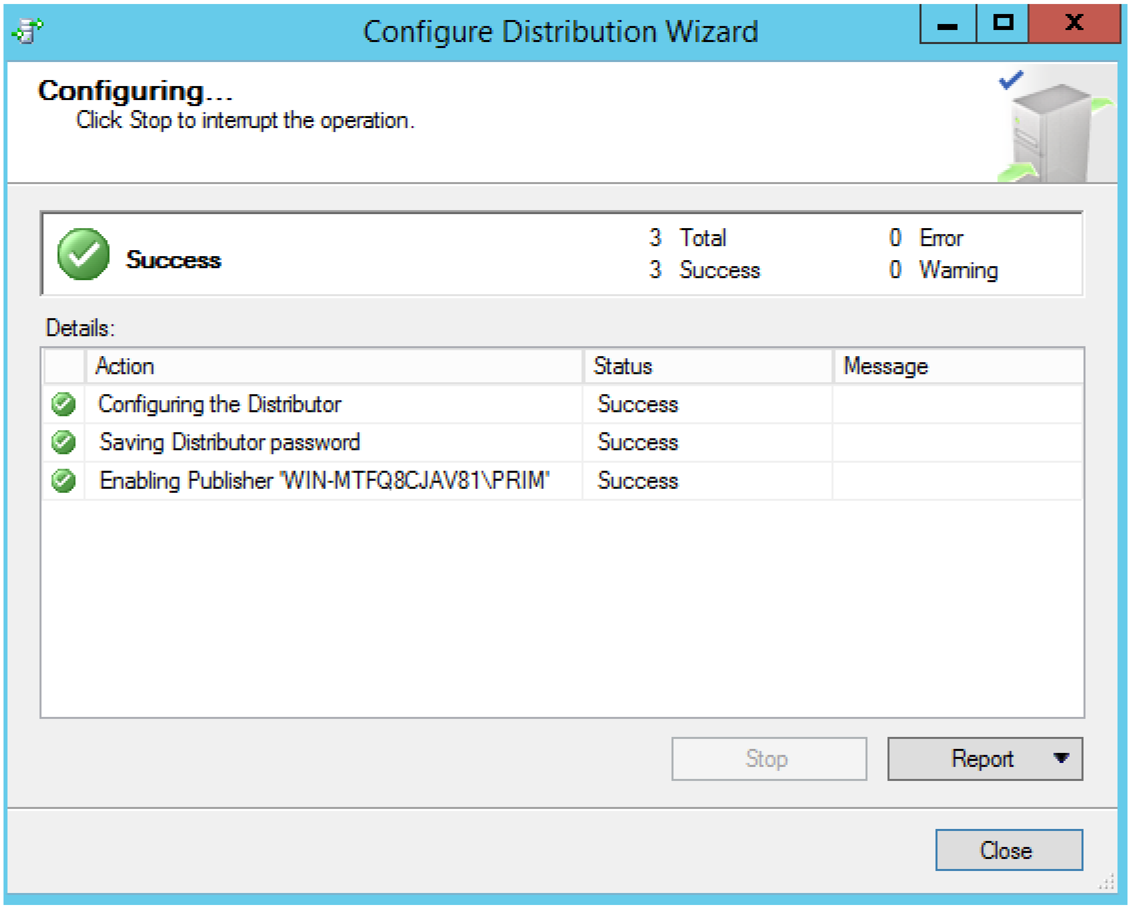
Enable servers to use this Distributor when they become Publishers.

Publishers:

	Publisher	Distribution Database
☑	WIN-MTFQ8CJAV81\PRIM	distribution
□	WIN-MTFQ8CJAV81\SECOND	

8. On the next panel you must specify a password that the Publishers using this distribution database will need to use. Specify the same password that you use for login. We will need this password later.

9. At the end of the process, you have configured the distributor successfully:



Configuring...
Click Stop to interrupt the operation.

Success

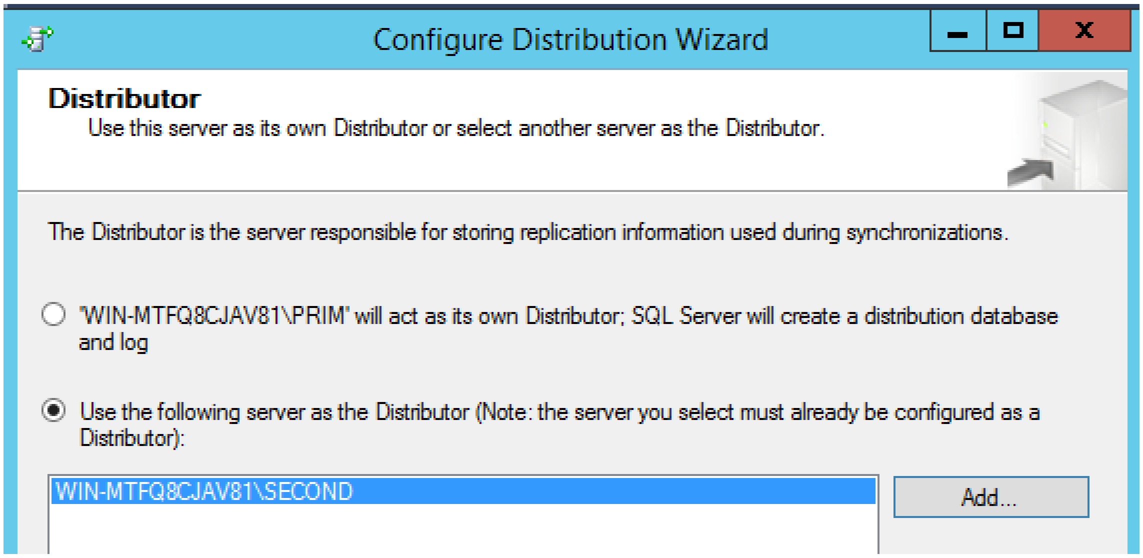
3 Total
3 Success
0 Error
0 Warning

Details:

	Action	Status	Message
✓	Configuring the Distributor	Success	
✓	Saving Distributor password	Success	
✓	Enabling Publisher 'WIN-MTFQ8CJAV81\PRIM'	Success	

Configuring the publisher

1. In order to configure the PRIM as publisher, we first disable it as distributor. Remember that so far the PRIM acted as its own distributor. In the pop-up menu of Replication select Disable Distribution and Publishing. When the PRIM has been disabled as a distributor, the pop-up menu changes. Select now Configure distribution and specify the SECOND instance as the distributor of PRIM:



Distributor

Use this server as its own Distributor or select another server as the Distributor.

The Distributor is the server responsible for storing replication information used during synchronizations.

○ 'WIN-MTFQ8CJAV81\PRIM' will act as its own Distributor; SQL Server will create a distribution database and log

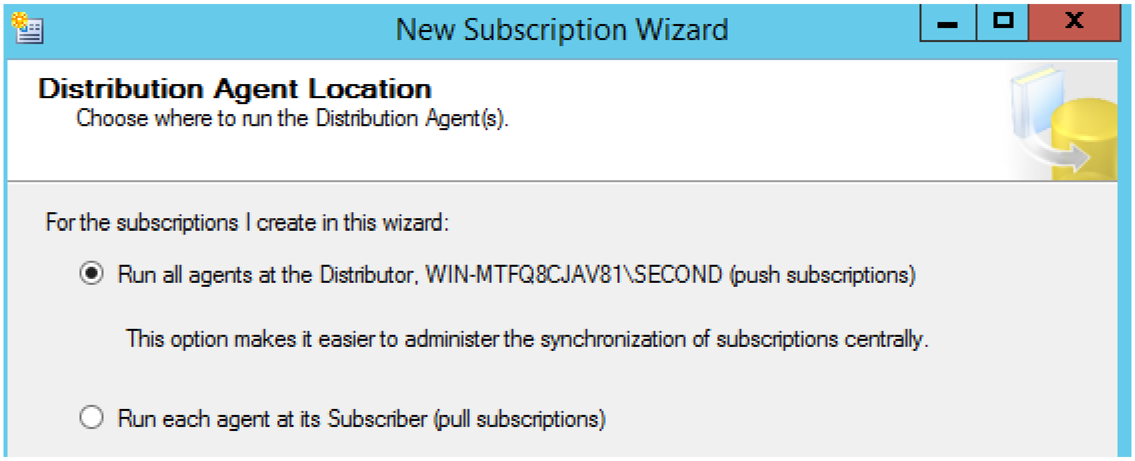
● Use the following server as the Distributor (Note: the server you select must already be configured as a Distributor):

WIN-MTFQ8CJAV81\SECOND

2. On the next pane, enter the same password as before (in Step 8).
3. The distribution is now set up.

Adding the publication and the subscription

1. Go on creating the transactional publication on PRIM in the usual way (Orders table without any filter).
2. We do not configure the Distribution properties on THIRD because THIRD will not act as a publisher.
3. Create a new subscription on the THIRD instance by selecting Add from Local subscriptions. Select PRIM as the publisher and select the publication that you created in the previous step.
4. Select Run all agents at the Distributor.



Distribution Agent Location

Choose where to run the Distribution Agent(s).

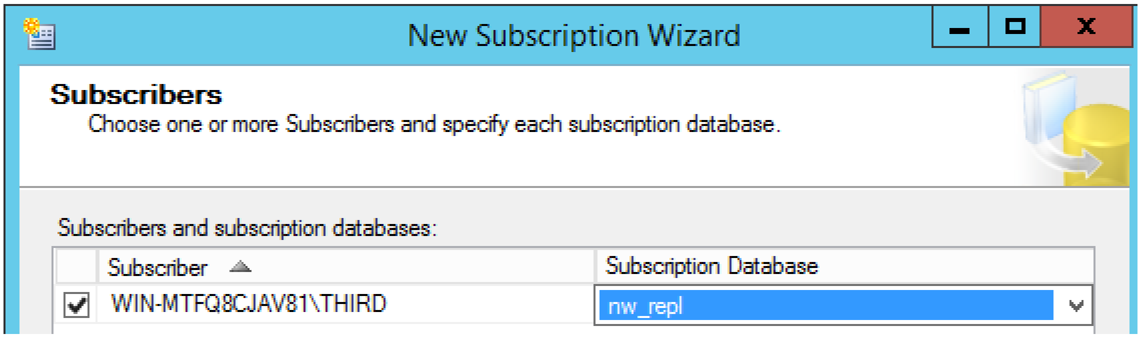
For the subscriptions I create in this wizard:

● Run all agents at the Distributor, WIN-MTFQ8CJAV81\SECOND (push subscriptions)

This option makes it easier to administer the synchronization of subscriptions centrally.

○ Run each agent at its Subscriber (pull subscriptions)

5. You can have the subscription database created on THIRD by the wizard as a new database named nw_repl.



Subscribers

Choose one or more Subscribers and specify each subscription database.

Subscribers and subscription databases:

	Subscriber	Subscription Database
☑	WIN-MTFQ8CJAV81\THIRD	nw_repl

6. Test the subscription. Open a query editor on the PRIM and update a record the Orders table. Open a query editor on the THIRD and verify that the change is propagated to the replicated table within 10 seconds.

7. Delete the publication at PRM and the subscription at THIRD. Delete the replicated table on THIRD.

PRACTICE: implement the loose coupling scenario for order event processing using transactional replication on the Products, Orders and Order details tables. Use the three servers in the above setup, with SECOND acting as the distributor. We suppose that the logistics division has its own database, running on the THIRD server.

1. Add a new field named 'status' to the Orders table in the northwind database of the PRIM instance with a default value of 0.

2. Change the status of existing orders from 0 to 2. We do not want to process all existing orders.

3. Replicate the three tables to the nw_repl database using transactional replication

4. Since the subscriber can change the replicated records and since the trading application using the Orders table will *never* update the status field, we will use this field on the subscriber to log the processing state of order records in a way similar to the case study solution. In this way we avoid using an extra log table. Details:

    a. We know that new orders will have a status of 0 by default. In order to mark changed orders, we can use an update trigger on the publisher that changes the status of the already existing record to 1

    b. The job at the subscriber processes order records with a status 0 or 1 and sets the status to 2 on success[5]

5. Implement and test the solution

6. Delete the publication and the subscription

Peer-to-Peer transactional replication

Allows writes to any of the nodes, changes are propagated to all the nodes automatically and only once. Designed for scale-out read load balancing. Concurrent writes of the same record at multiple nodes i.e. write conflicts are not allowed and treated as a critical error that requires manual resolution[6].

[5] This will work fine if the order is updated only once on the publisher side.

[6] https://learn.microsoft.com/en-us/sql/relational-databases/replication/transactional/peer-to-peer-transactional-replication

Merge replication

Scenario: our sales employees are traveling and making new orders for their clients. Occasionally they also need to change previous orders when, for example, a shipping address changes. It may also happen that two employees modify the same order. The employees are not continuously connected to the internet. We must design a replication solution that merges all such changes with each other and with the central Northwind database.

In **merge replication**, the duty of the log reader agent is taken by triggers, tables, and views which are created automatically in each subscriber database and also in the publisher database. The triggers log changes in special system tables named 'MSmerge_*' that are created in the same database as the replicated table. There are three triggers named 'MSmerge_[ins, upd, del]_*' created for each table. There are also database level *schema triggers* that log the changes in the schema of the replicated tables.

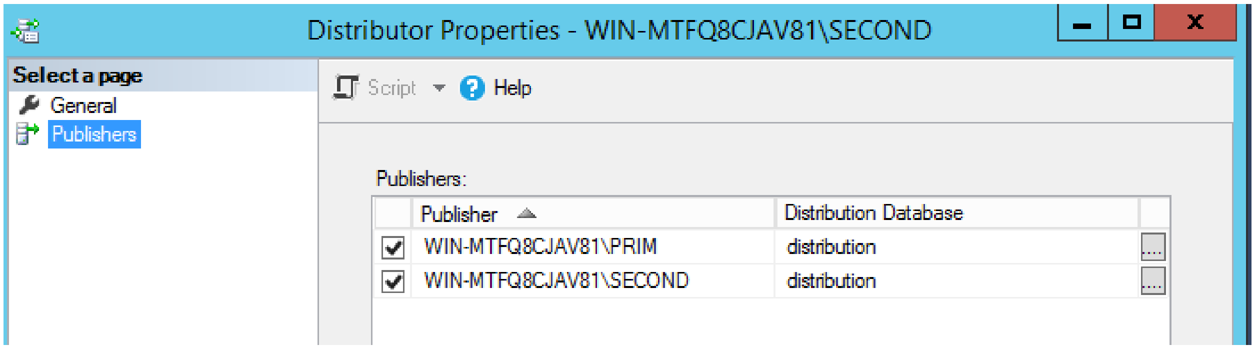
Merge replication starts with an initial snapshot created by a snapshot agent. By default, the snapshot agent is configured to run every 14 days. Then the duty of the merge agent is similar to the distribution agent in transactional replication with the difference that merge replication is by default configured bi-directional. This means that the agent applies changes on the subscriber and publisher sides as well. There is a separate merge agent for each subscription. The merge agent runs on the distributor in case of a push distribution and on the subscriber in case of a pull subscription.

In order to support the bi-directional data synchronization, articles in merge replication must have a uniqueidentifier (GUID) type column that is similar to the identity data type but it produces globally unique IDs[7]. If such a column does not exist, it is added automatically to the tables—which can potentially crash the legacy applications already using the table. The new field is remover if the publication is deleted.

The publication

PRACTICE: In order to develop the solution, follow the steps below.

1. Set the SECOND instance as publisher even if it will not publish any publications:



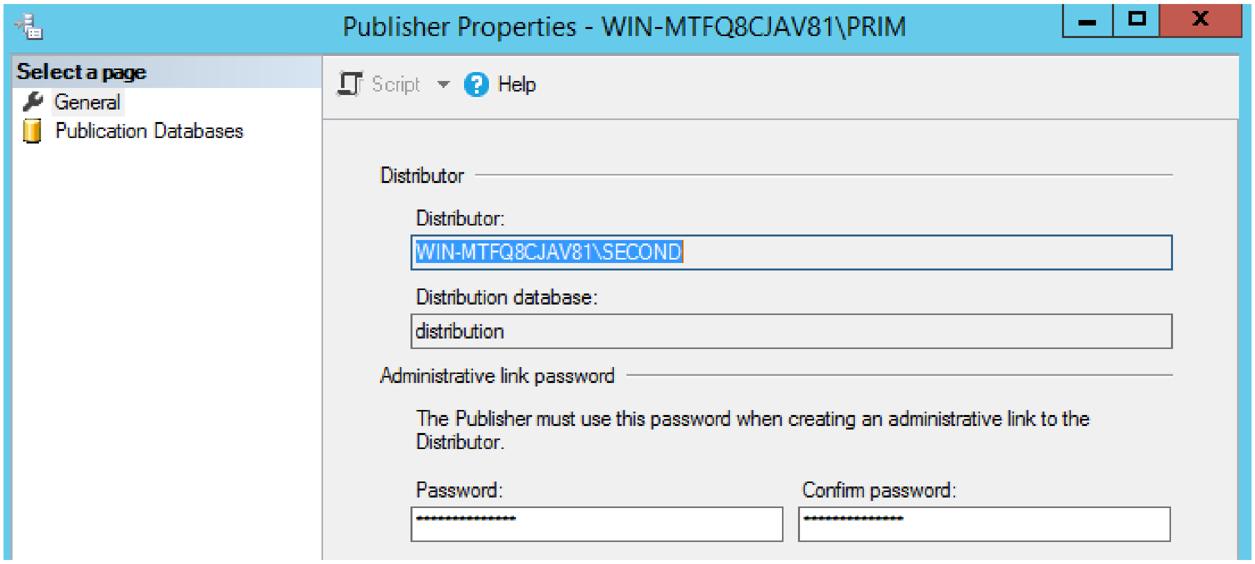
Distributor Properties - WIN-MTFQ8CJAV81\SECOND

Publishers:

	Publisher	Distribution Database
☑	WIN-MTFQ8CJAV81\PRIM	distribution
☑	WIN-MTFQ8CJAV81\SECOND	distribution

2. Configure the SECOND instance as the distributor of the PRIM (see the previous section). After that, this is what you should see on the PRIM:

[7] Use this data type like this: CREATE TABLE test (my_guid uniqueidentifier DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,… etc



Distributor

Distributor: WIN-MTFQ8CJAV81\SECOND

Distribution database: distribution

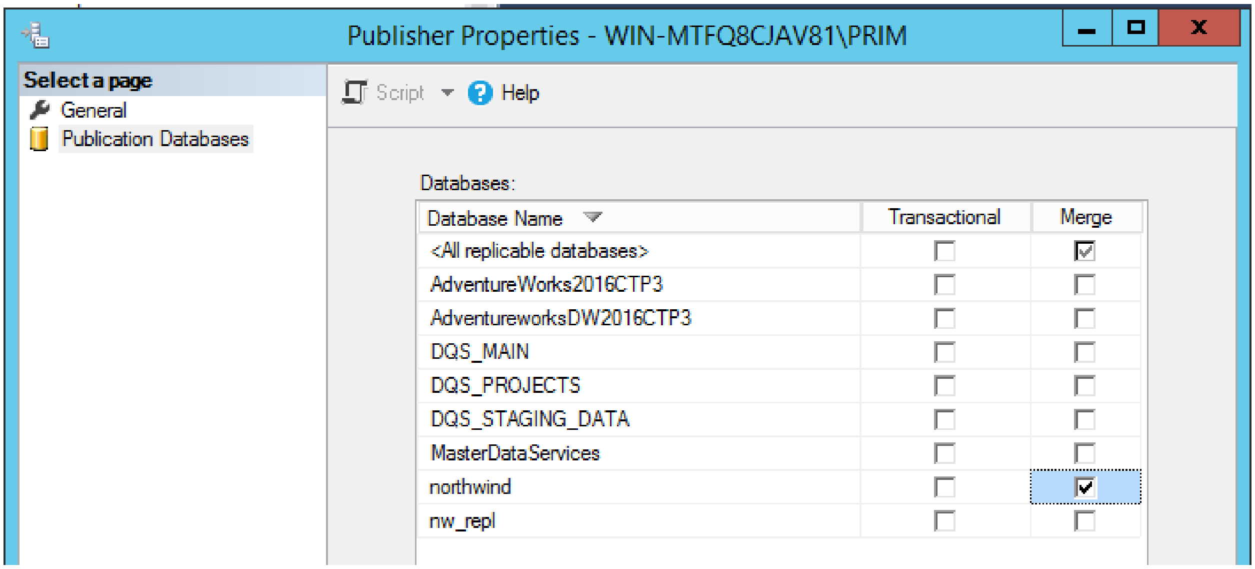
Administrative link password

The Publisher must use this password when creating an administrative link to the Distributor.

Password: ***************

Confirm password: ***************

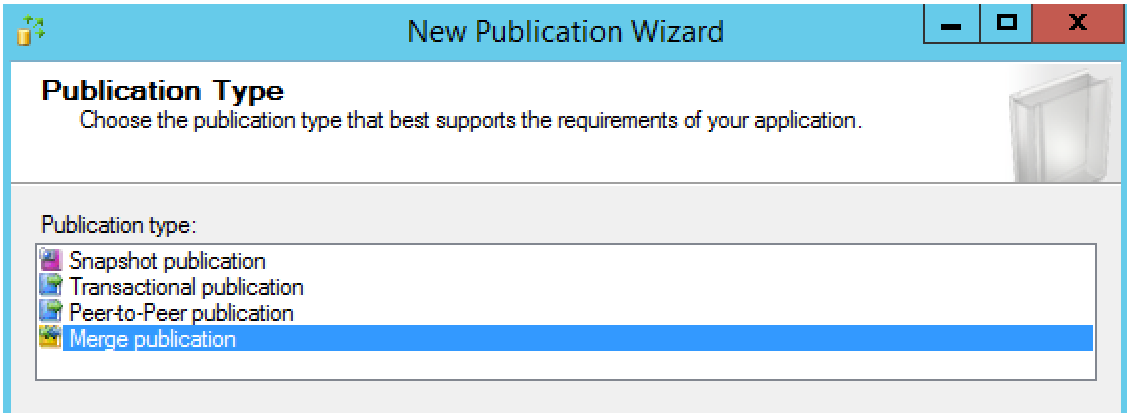
3. Enable the northwind database for merge replication. Select the Publication Databases tab on the publisher (PRIM):



Databases:

Database Name	Transactional	Merge
<All replicable databases>	□	☑
AdventureWorks2016CTP3	□	□
AdventureworksDW2016CTP3	□	□
DQS_MAIN	□	□
DQS_PROJECTS	□	□
DQS_STAGING_DATA	□	□
MasterDataServices	□	□
northwind	□	☑
nw_repl	□	□

4. Select the northwind database for the publication database and select the publication type:



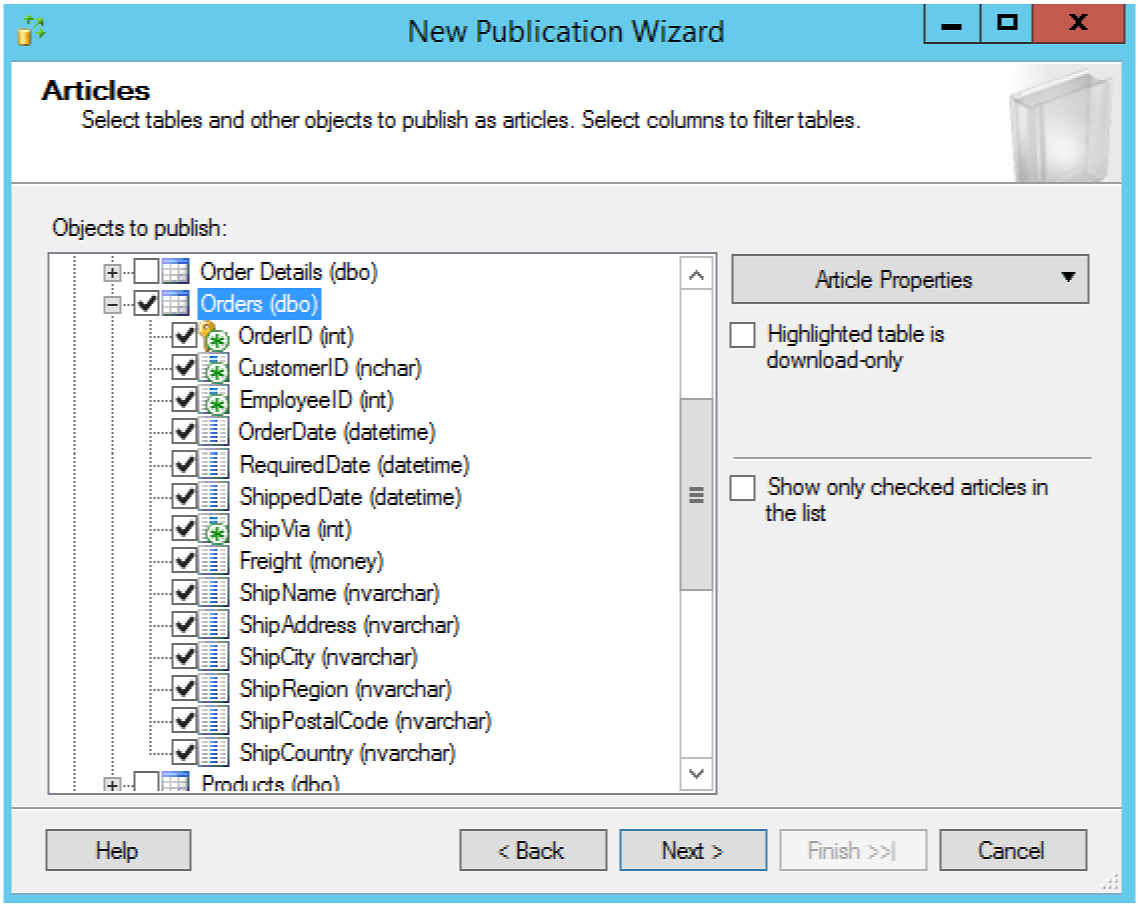
Publication Type

Choose the publication type that best supports the requirements of your application.

Publication type:

- Snapshot publication
- Transactional publication
- Peer-to-Peer publication
- Merge publication

5. Accept the default subscriber types and select the Orders table as the single article of the publication.



Articles

Select tables and other objects to publish as articles. Select columns to filter tables.

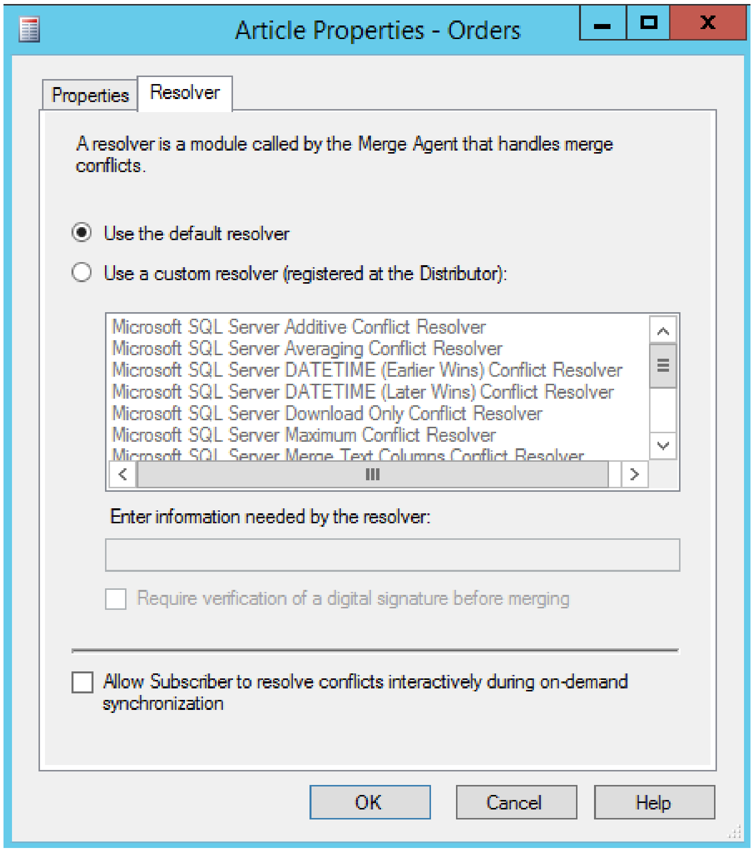
Objects to publish:

- □ Order Details (dbo)
- ☑ Orders (dbo)
  - ☑ OrderID (int)
  - ☑ CustomerID (nchar)
  - ☑ EmployeeID (int)
  - ☑ OrderDate (datetime)
  - ☑ RequiredDate (datetime)
  - ☑ ShippedDate (datetime)
  - ☑ ShipVia (int)
  - ☑ Freight (money)
  - ☑ ShipName (nvarchar)
  - ☑ ShipAddress (nvarchar)
  - ☑ ShipCity (nvarchar)
  - ☑ ShipRegion (nvarchar)
  - ☑ ShipPostalCode (nvarchar)
  - ☑ ShipCountry (nvarchar)
- □ Products (dbo)

□ Highlighted table is download-only

□ Show only checked articles in the list

6. You can set various properties for each article by selecting Article Properties, including the resolver to be used for various conflict types. A conflict occurs when an agent tries to change a record that has a pending change. The built-in resolver modules currently registered at the server can be listed by the sp_enumcustomresolvers procedure. For the required parameters of the built-in resolvers, see[8]. You could also add your own resolver as a stored procedure or DLL[9]. The default resolver implements the 'first to Publisher wins' strategy:
    a. if the conflict occurs between a Publisher and a Subscriber, the Publisher change is accepted and the Subscriber value is refused.
    b. if the conflict occurs between two Subscribers using pull subscriptions, then the change from the first Subscriber to synchronize with the Publisher is accepted, and others are refused.



A resolver is a module called by the Merge Agent that handles merge conflicts.

● Use the default resolver

○ Use a custom resolver (registered at the Distributor):

Microsoft SQL Server Additive Conflict Resolver
Microsoft SQL Server Averaging Conflict Resolver
Microsoft SQL Server DATETIME (Earlier Wins) Conflict Resolver
Microsoft SQL Server DATETIME (Later Wins) Conflict Resolver
Microsoft SQL Server Download Only Conflict Resolver
Microsoft SQL Server Maximum Conflict Resolver
Microsoft SQL Server Merge Text Columns Conflict Resolver

Enter information needed by the resolver:

□ Require verification of a digital signature before merging

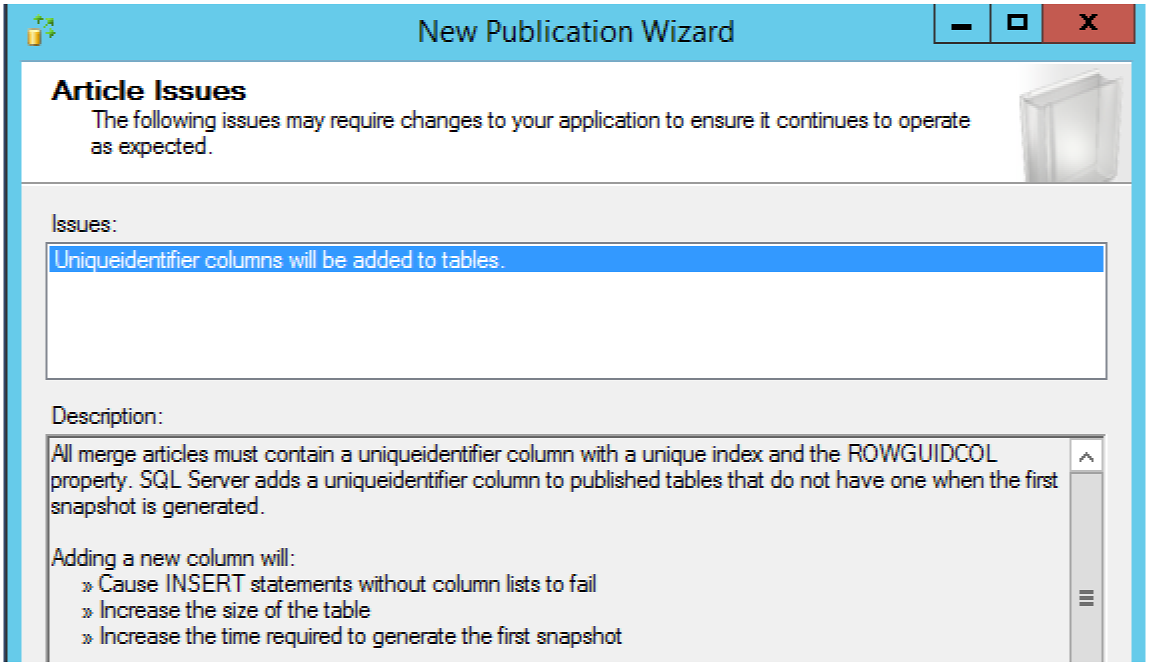
□ Allow Subscriber to resolve conflicts interactively during on-demand synchronization

7. The next panel warns that a new GUID will be added to the table. This does not change the primary and foreign key constraints of the table.

[8] https://learn.microsoft.com/en-us/sql/relational-databases/replication/merge/advanced-merge-replication-conflict-com-based-resolvers?view=sql-server-ver16

[9] https://learn.microsoft.com/en-us/sql/relational-databases/replication/implement-a-custom-conflict-resolver-for-a-merge-article

The custom resolver procedure is run on the Publisher and it is called by the Merge agent. It queries the changed record from the Subscriber using the row GUID received from the Merge agent, and returns a result set with a single record that has the same fields as the base table and that contains the values for the winning version of the record.



Article Issues

The following issues may require changes to your application to ensure it continues to operate as expected.

Issues:

Uniqueidentifier columns will be added to tables.

Description:

All merge articles must contain a uniqueidentifier column with a unique index and the ROWGUIDCOL property. SQL Server adds a uniqueidentifier column to published tables that do not have one when the first snapshot is generated.

Adding a new column will:
- Cause INSERT statements without column lists to fail
- Increase the size of the table
- Increase the time required to generate the first snapshot

8. For the snapshot agent scheduling, accept the defaults, then define the agent security in the usual way.

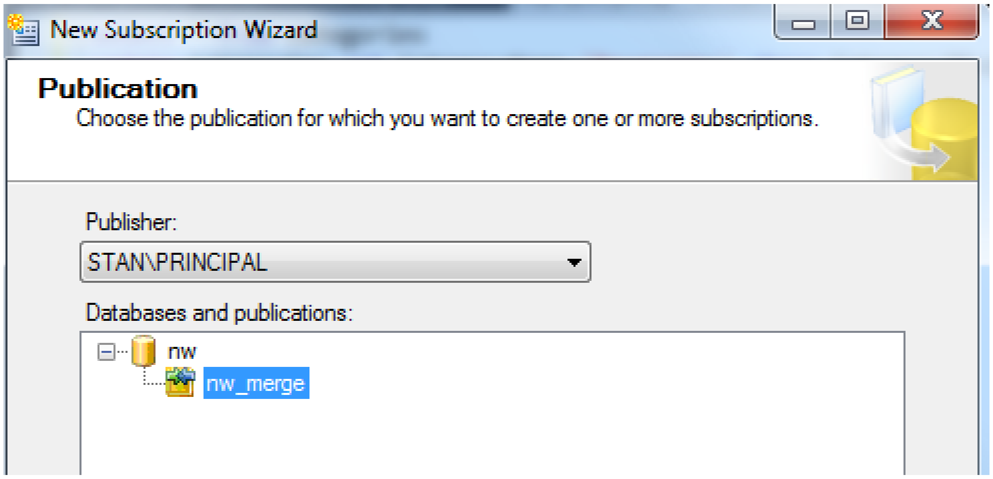
9. Name the publication nw_merge and create it. Check the new GUID column and the 3 new triggers on the Orders table.

The subscription

We add two subscribers to see the merge process in action.

1. Start the SQL Server Agent on THIRD manually

2. On the THIRD instance, select New subscription and specify the PRIM as the publisher:



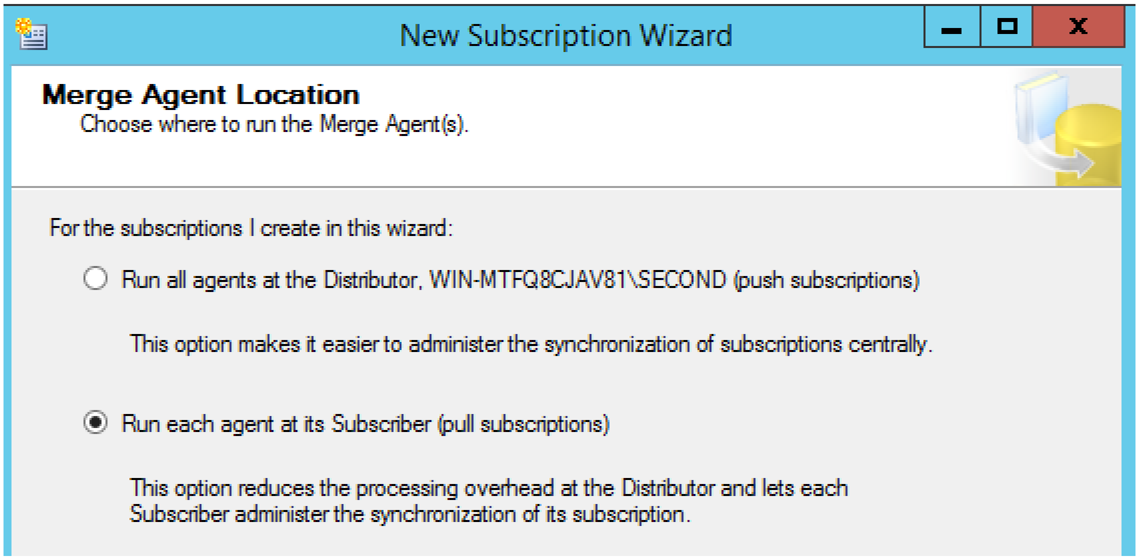
Publication

Choose the publication for which you want to create one or more subscriptions.

Publisher: STAN\PRINCIPAL

Databases and publications:

3. On the next pane, select pull subscription. This is in line with the usual expectation that the subscribers will want to schedule their synchronization.



Merge Agent Location

Choose where to run the Merge Agent(s).

For the subscriptions I create in this wizard:

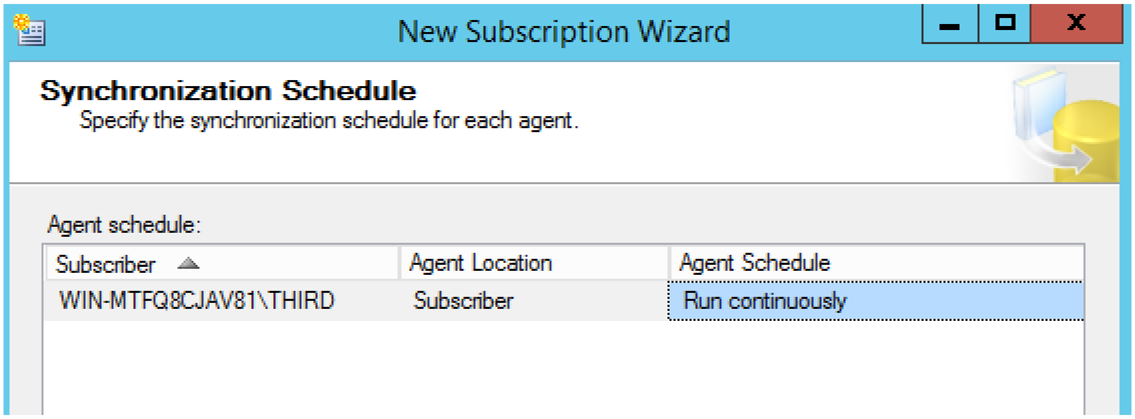
○ Run all agents at the Distributor, WIN-MTFQ8CJAV81\SECOND (push subscriptions)

This option makes it easier to administer the synchronization of subscriptions centrally.

● Run each agent at its Subscriber (pull subscriptions)

This option reduces the processing overhead at the Distributor and lets each Subscriber administer the synchronization of its subscription.

4. Add the two subscribers, one on the THIRD instance and one on the PRIM, with newly created databases named nw_merge_1-2.
5. Set the security on the agents in the usual way.
6. We set the synch schedule for both merge agents to 'Run continuously'. *Note: this setting is only for test and demo purposes. In a real life scenario, the merge process would start either at specified intervals or on demand, when a specific event, e.g. a VPN connection occurs on the subscriber.*



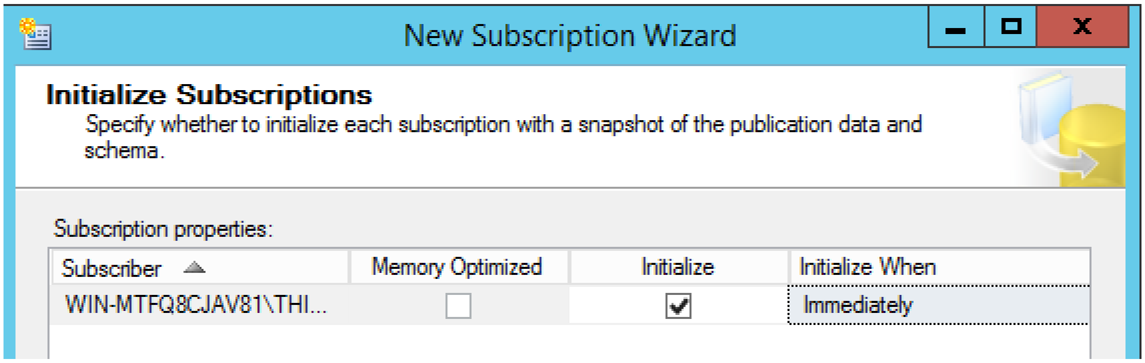
Synchronization Schedule

Specify the synchronization schedule for each agent.

Agent schedule:

Subscriber	Agent Location	Agent Schedule
WIN-MTFQ8CJAV81\THIRD	Subscriber	Run continuously

7. Initialize the subscriptions immediately.



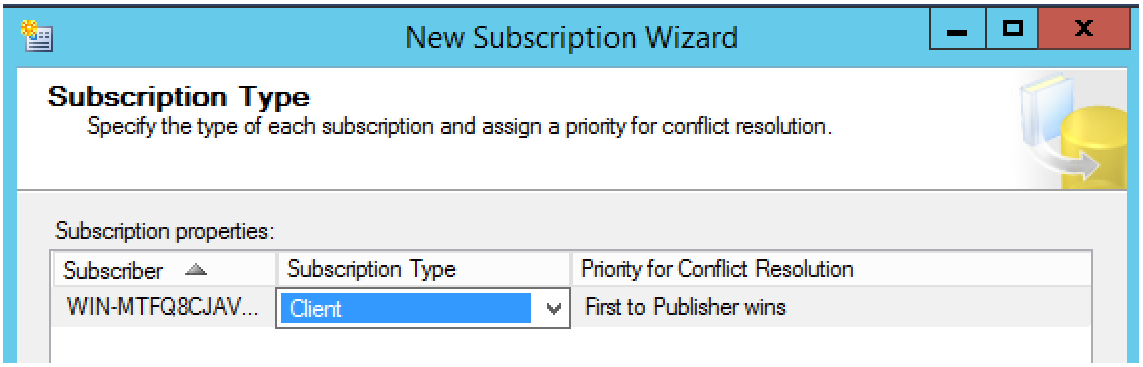
Initialize Subscriptions

Specify whether to initialize each subscription with a snapshot of the publication data and schema.

Subscription properties:

Subscriber	Memory Optimized	Initialize	Initialize When
WIN-MTFQ8CJAV81\THI...	□	☑	Immediately

8. In merge publication, the subscribers may be allowed to republish the publication to which they are subscribed (Server type subscription), thus creating a hierarchical subscribing architecture. Since we do not want to create a hierarchy of subscribers, we select Client type—no republication.



Subscription Type

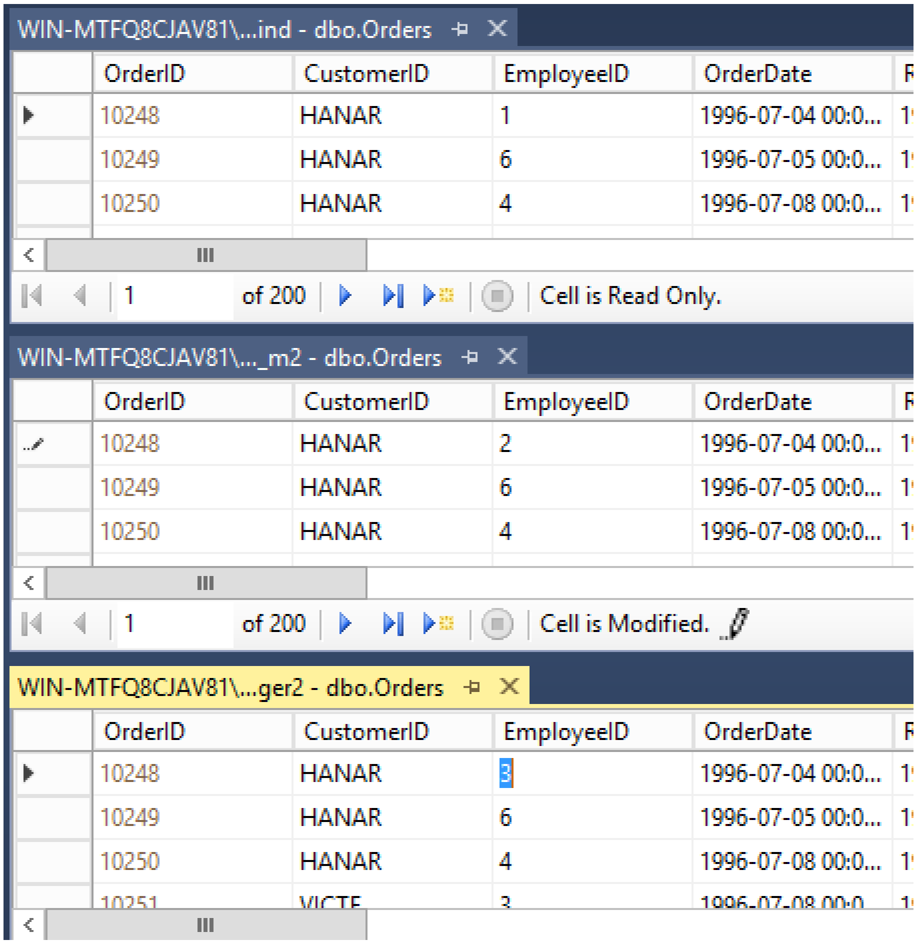
Specify the type of each subscription and assign a priority for conflict resolution.

Subscription properties:

Subscriber	Subscription Type	Priority for Conflict Resolution
WIN-MTFQ8CJAV...	Client	First to Publisher wins

9. Finalize and start the replication.

10. Verify that the changed values are propagated from any subscriber or the publisher to all the other by editing the Orders table in three separate editors. Be patient: It can take up to 2 minutes for the synchronization to complete. Note the first-come-first-served style default conflict resolution when the two subscribers update a record ‘simultaneously’. The change request that reaches first the distribution database will be applied to all parties. Use the Window -> Add horizontal tab group command in SSMS to see all three tables simultaneously. You can refresh the data grids by pressing Ctl-R.



WIN-MTFQ8CJAV81\...ind - dbo.Orders

	OrderID	CustomerID	EmployeeID	OrderDate	R
▶	10248	HANAR	1	1996-07-04 00:0...	1
	10249	HANAR	6	1996-07-05 00:0...	1
	10250	HANAR	4	1996-07-08 00:0...	1

1 of 200 Cell is Read Only.

WIN-MTFQ8CJAV81\..._m2 - dbo.Orders

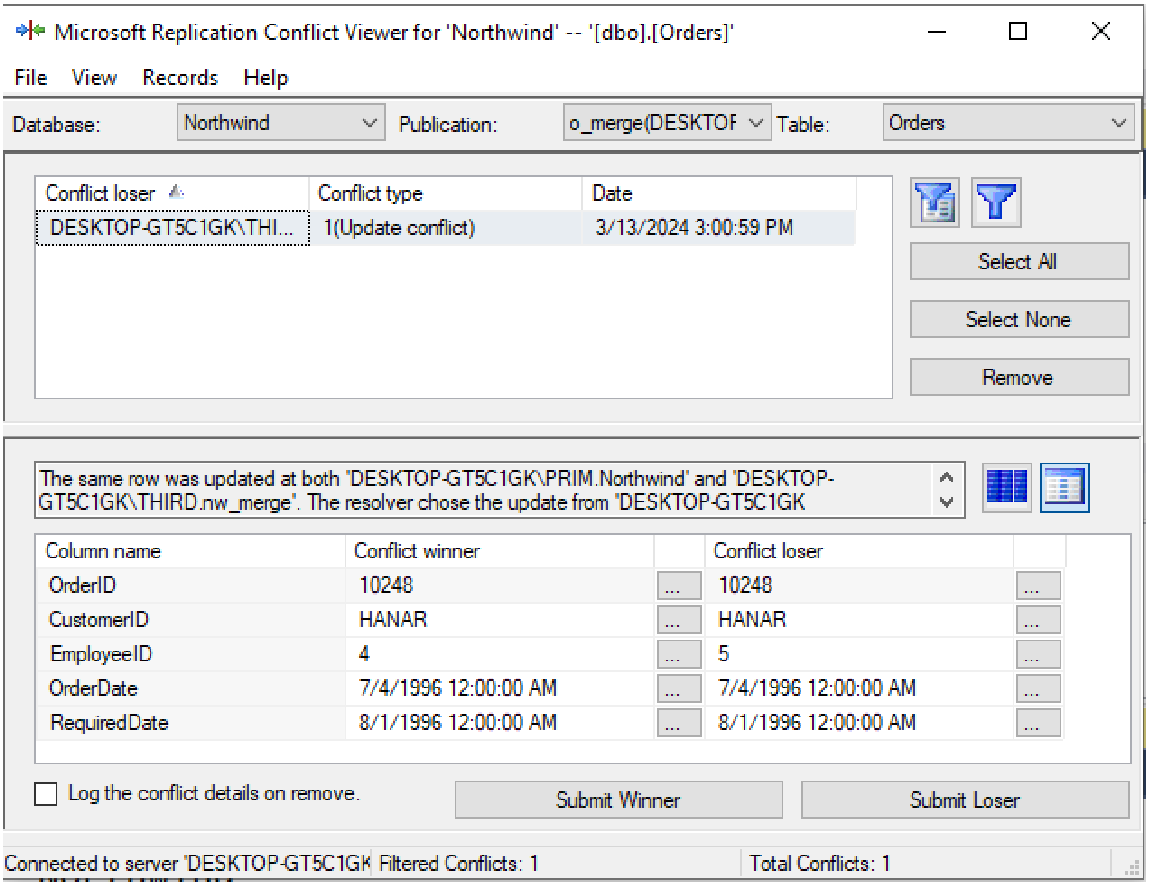
	OrderID	CustomerID	EmployeeID	OrderDate	R
✎	10248	HANAR	2	1996-07-04 00:0...	1
	10249	HANAR	6	1996-07-05 00:0...	1
	10250	HANAR	4	1996-07-08 00:0...	1

1 of 200 Cell is Modified.

WIN-MTFQ8CJAV81\...ger2 - dbo.Orders

	OrderID	CustomerID	EmployeeID	OrderDate	R
▶	10248	HANAR	3	1996-07-04 00:0...	1
	10249	HANAR	6	1996-07-05 00:0...	1
	10250	HANAR	4	1996-07-08 00:0...	1
	10251	VICTE	3	1996-07-08 00:0	1

11. Merge conflicts can be viewed and manually resolved in the Replication Conflict Viewer tool, which is available from the pop-up menu of the publication. If a record has been updated by all three servers i.e. the conflict involves 3 parties, it will appear as two separate conflicts, with the same winner. Here you can either accept the default resolution (by clicking Submit Winner) or revert it (by clicking Submit Loser):


Microsoft Replication Conflict Viewer for 'Northwind' -- '[dbo].[Orders]'
File
View
Records
Help
Select All
Select None
Remove
Submit Winner
Submit Loser

Database: Northwind Publication: o_merge(DESKTOF Table: Orders

Conflict loser	Conflict type	Date
DESKTOP-GT5C1GK\THI...	1(Update conflict)	3/13/2024 3:00:59 PM

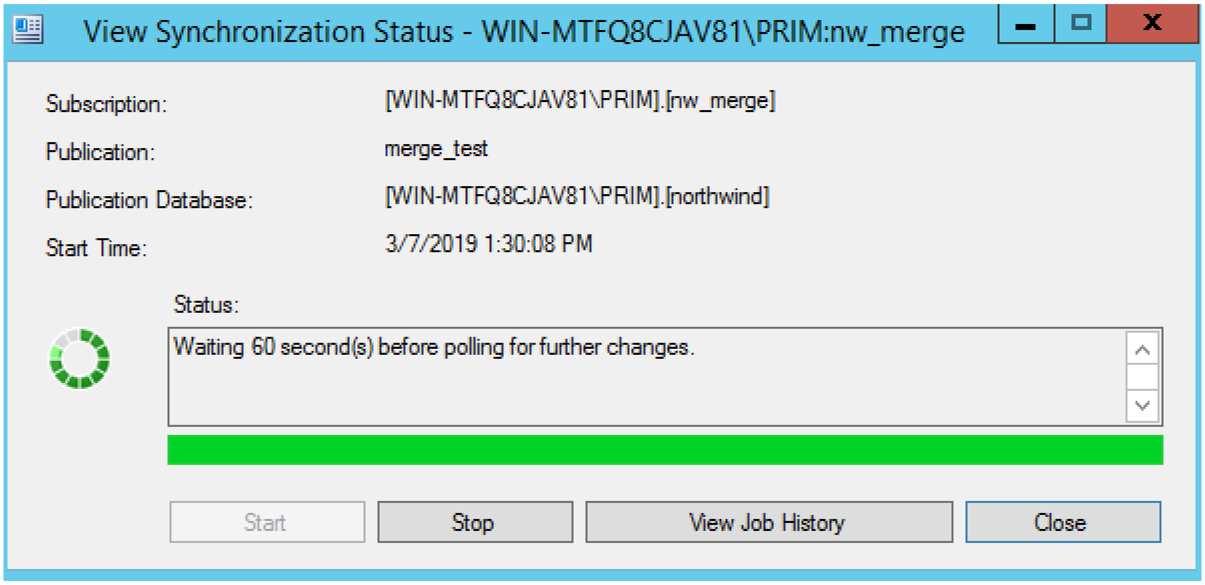
The same row was updated at both 'DESKTOP-GT5C1GK\PRIM.Northwind' and 'DESKTOP-GT5C1GK\THIRD.nw_merge'. The resolver chose the update from 'DESKTOP-GT5C1GK

Column name	Conflict winner	Conflict loser
OrderID	10248	10248
CustomerID	HANAR	HANAR
EmployeeID	4	5
OrderDate	7/4/1996 12:00:00 AM	7/4/1996 12:00:00 AM
RequiredDate	8/1/1996 12:00:00 AM	8/1/1996 12:00:00 AM

☐ Log the conflict details on remove.

Connected to server 'DESKTOP-GT5C1GK Filtered Conflicts: 1 Total Conflicts: 1

12. You can review the current state of the merge process by selecting View synch status from the pop-up menu of the subscription. You can also start the process manually.



Subscription: [WIN-MTFQ8CJAV81\PRIM].[nw_merge]

Publication: merge_test

Publication Database: [WIN-MTFQ8CJAV81\PRIM].[northwind]

Start Time: 3/7/2019 1:30:08 PM

Status:

Waiting 60 second(s) before polling for further changes.

13. Generate and review replication scripts on both instances by selecting Generate scripts from the pop-up menu of the Replication group.

14. Delete all replication objects.

PRACTICE: In the new scenario, the employees are updating the UnitPrice of the Products table in the Northwind database and merging concurrent changes. One of the employees, using the THIRD server, has lower priority than the other. Set up a merge publication from PRIM with pull subscriptions on SECOND and THIRD. Use Server type subscription to set the priority. Use the Conflict viewer to verify that the employee with the lower priority will always lose, regardless of the timing.

Log shipping

While the primary application area of replication is business process automation, log shipping is designed to prepare for **disaster recovery** of a database, by creating and synchronizing an exact, usually read-only copy of the database also known as a 'warm' standby database. A log can be shipped from the primary sever to multiple secondary servers.

Log shipping starts with a full backup of the database which is restored on the client. Then the three steps involved in log shipping are as follows.

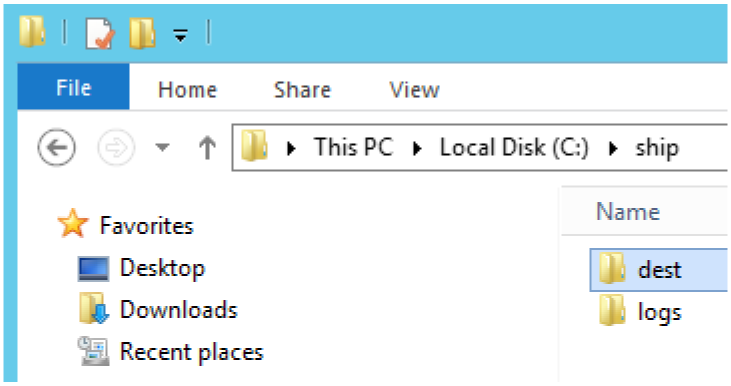
1. A backup job running on the primary server backs up the new portion of the database transaction log to the local server
2. A copy job running on the secondary server copies the log to a configurable destination (e.g. a network file server)
3. A restore job running on the secondary server restores the backup to the secondary database(s)

An alert job may also be running monitoring whether each step is performed as expected and on time.

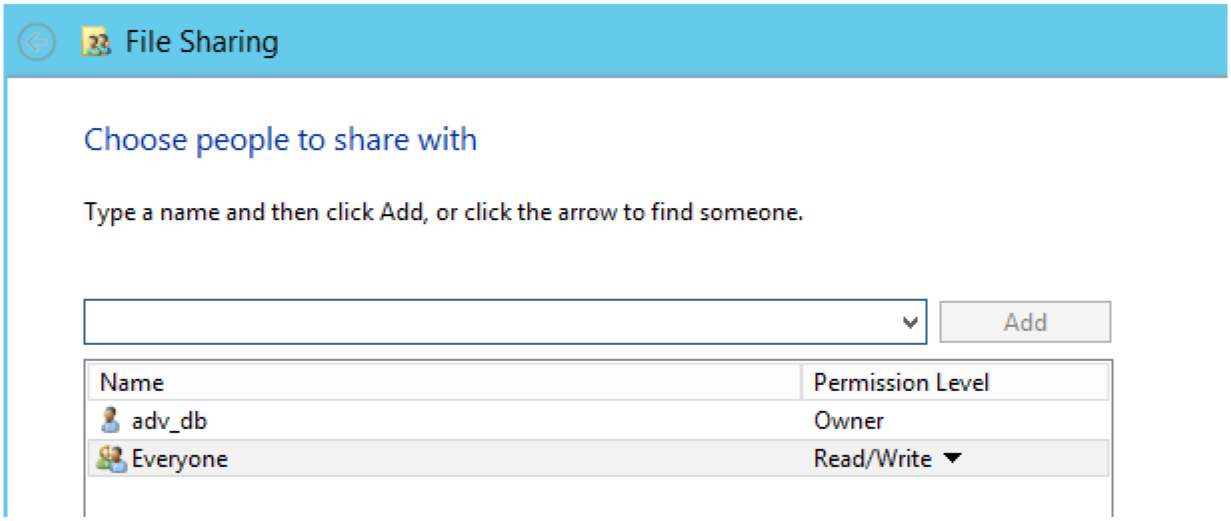
Depending on the scheduling of the jobs, there is a latency between the two databases. This latency can be exploited when the primary database is modified by mistake.

PRACTICE: In our scenario, we create a '**warm backup'** of the Northwind database as follows. This means that in case of a server failure, the standby server (i.e. the warm backup) can replace the production server in a couple of minutes, without a long full restore procedure. The primary and secondary servers run normally on different server machines, but in our demo we will use the same machine and two server instances, the Primary and the Secondary.

1. Create a folder for the storing of the logs and another where the copy will be placed.



2. Set sharing for *both* folders to shared to everyone (Properties/Sharing/Share). Type Everyone and set Read/Write.

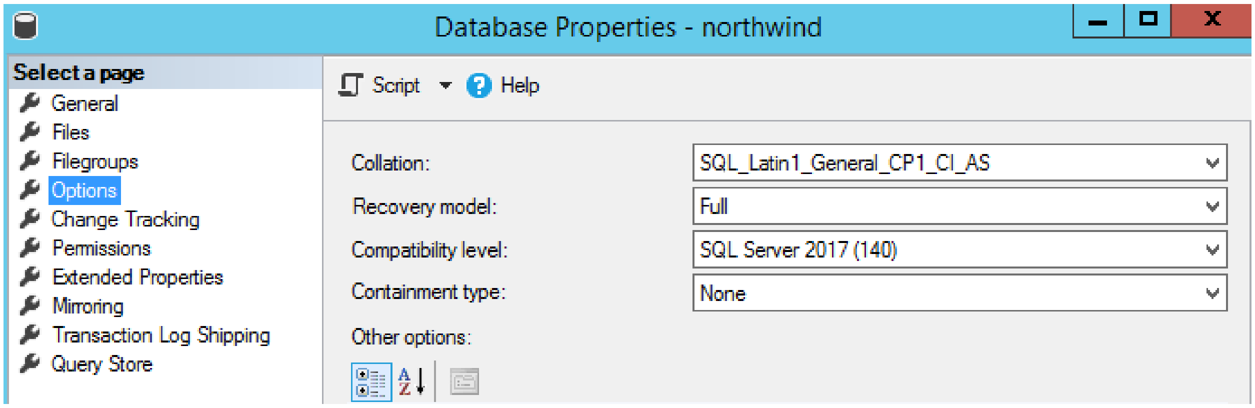


Choose people to share with

Type a name and then click Add, or click the arrow to find someone.

Name	Permission Level
adv_db	Owner
Everyone	Read/Write

3. In the pop-up menu of the Northwind database on the Primary server, select Properties and verify that the recovery model of the Northwind database is set to Full. This means that the inactive parts of the log will not be deleted at a checkpoint.



Collation: SQL_Latin1_General_CP1_CI_AS

Recovery model: Full

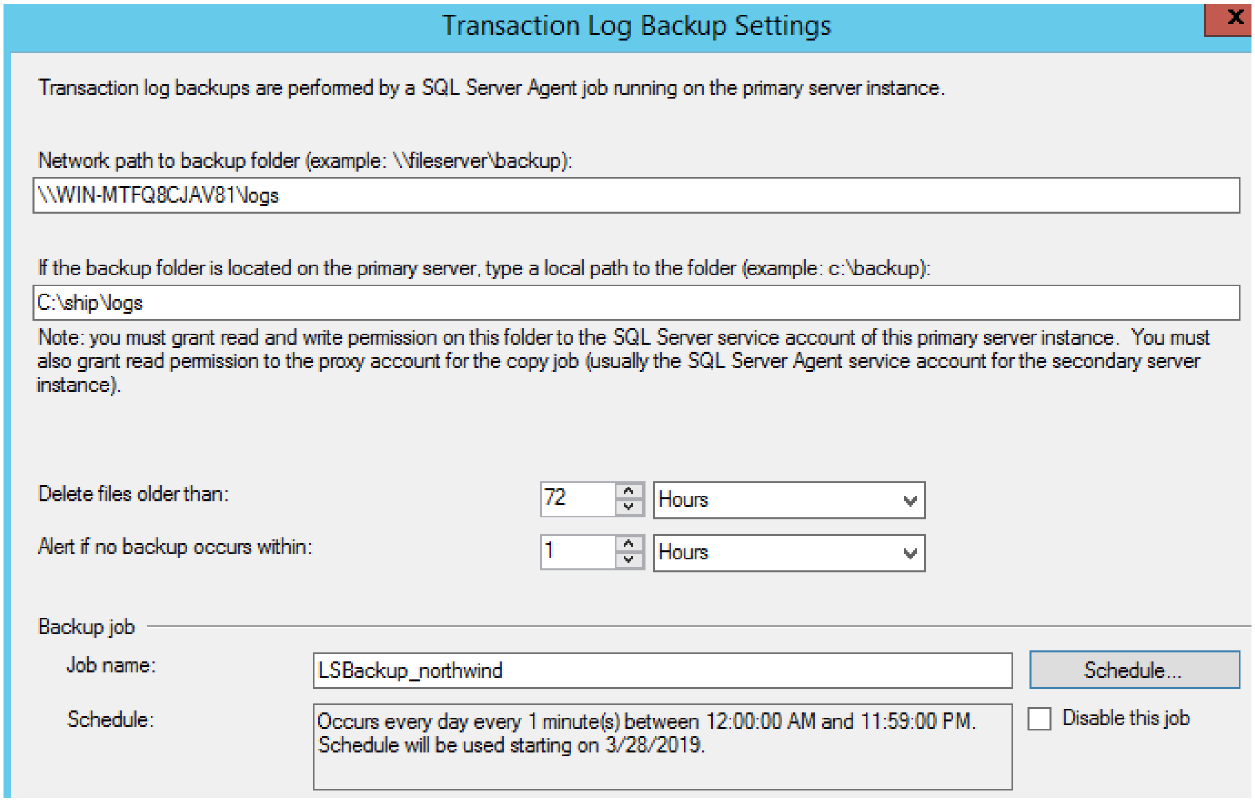
Compatibility level: SQL Server 2017 (140)

Containment type: None

Other options:

4. On the PRIM, in the Nortwind database Tasks -> Ship Transaction logs panel, enable the database for shipping, and select Backup settings. Type \\WIN-MTFQ8CJAV81\logs as the network path of the backup folder and C:\ship\logs as the local folder path.

5. Select Edit job and set a schedule that runs the job every minute. *Note: this setting is only for demo purposes.*



Transaction Log Backup Settings

Transaction log backups are performed by a SQL Server Agent job running on the primary server instance.

Network path to backup folder (example: \\fileserver\backup):
\\WIN-MTFQ8CJAV81\logs

If the backup folder is located on the primary server, type a local path to the folder (example: c:\backup):
C:\ship\logs

Note: you must grant read and write permission on this folder to the SQL Server service account of this primary server instance. You must also grant read permission to the proxy account for the copy job (usually the SQL Server Agent service account for the secondary server instance).

Delete files older than: 72 Hours

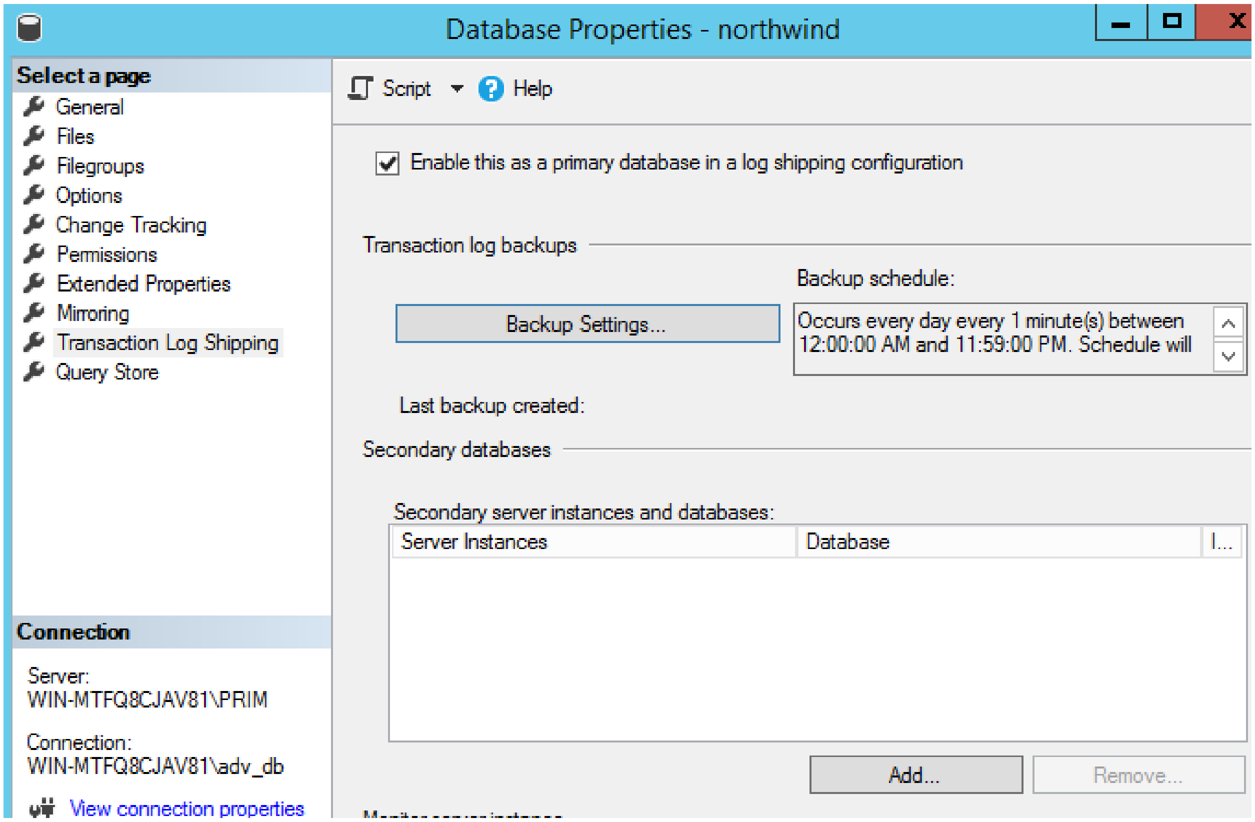
Alert if no backup occurs within: 1 Hours

Backup job

Job name: LSBackup_northwind

Schedule: Occurs every day every 1 minute(s) between 12:00:00 AM and 11:59:00 PM. Schedule will be used starting on 3/28/2019.

6. Return to the Database properties panel and select Add in the Secondary databases section:



Database Properties - northwind

☑ Enable this as a primary database in a log shipping configuration

Transaction log backups

Backup schedule:

Occurs every day every 1 minute(s) between 12:00:00 AM and 11:59:00 PM. Schedule will

Last backup created:

Secondary databases

Secondary server instances and databases:

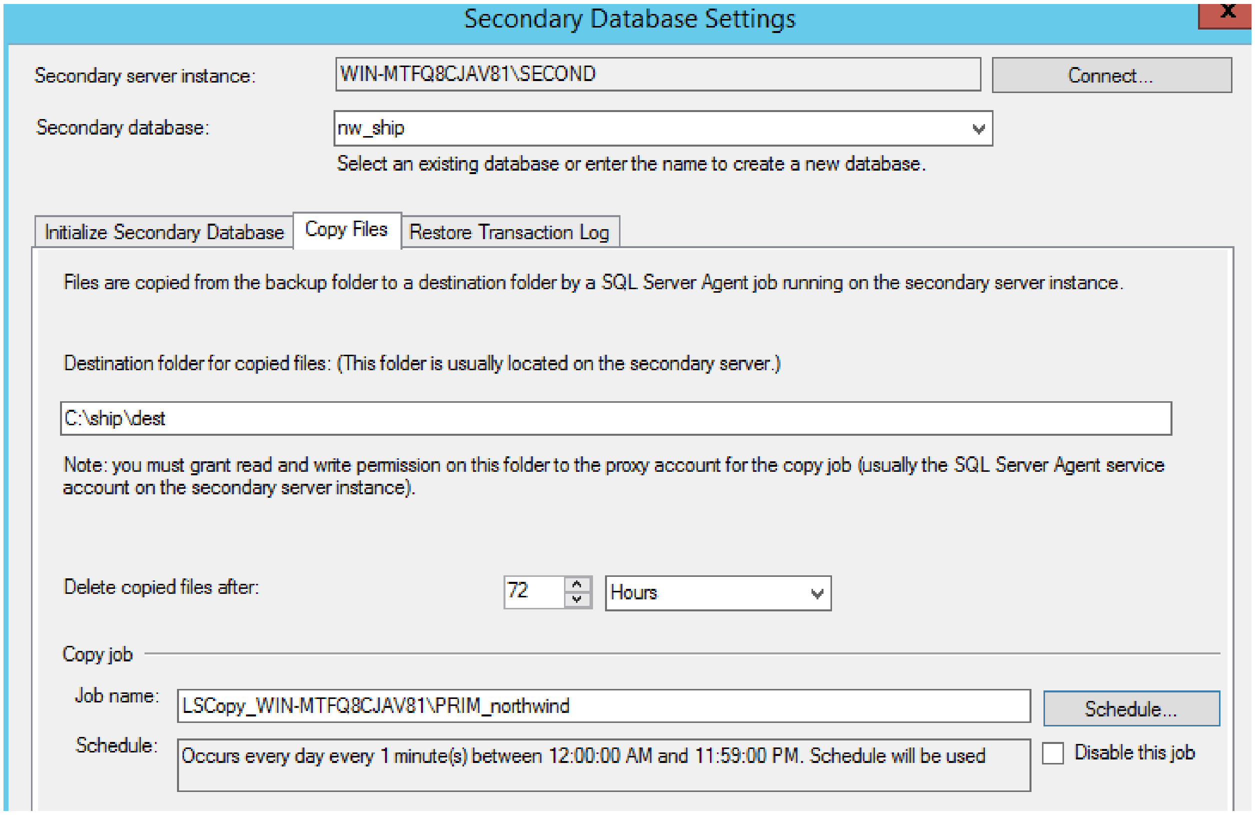
Server Instances	Database	I...

Connection

Server:
WIN-MTFQ8CJAV81\PRIM

Connection:
WIN-MTFQ8CJAV81\adv_db

7. On the Secondary database settings panel, type the new database name nw_ship. Accept the default for initialization. In the Copy files tab, enter C:\ship\dest as the destination path. Set the schedule for every 1 minute. *Note: this setting is only for demo purposes*.



Secondary Database Settings

Secondary server instance: WIN-MTFQ8CJAV81\SECOND

Secondary database: nw_ship

Select an existing database or enter the name to create a new database.

Files are copied from the backup folder to a destination folder by a SQL Server Agent job running on the secondary server instance.

Destination folder for copied files: (This folder is usually located on the secondary server.)

C:\ship\dest

Note: you must grant read and write permission on this folder to the proxy account for the copy job (usually the SQL Server Agent service account on the secondary server instance).

Delete copied files after: 72 Hours

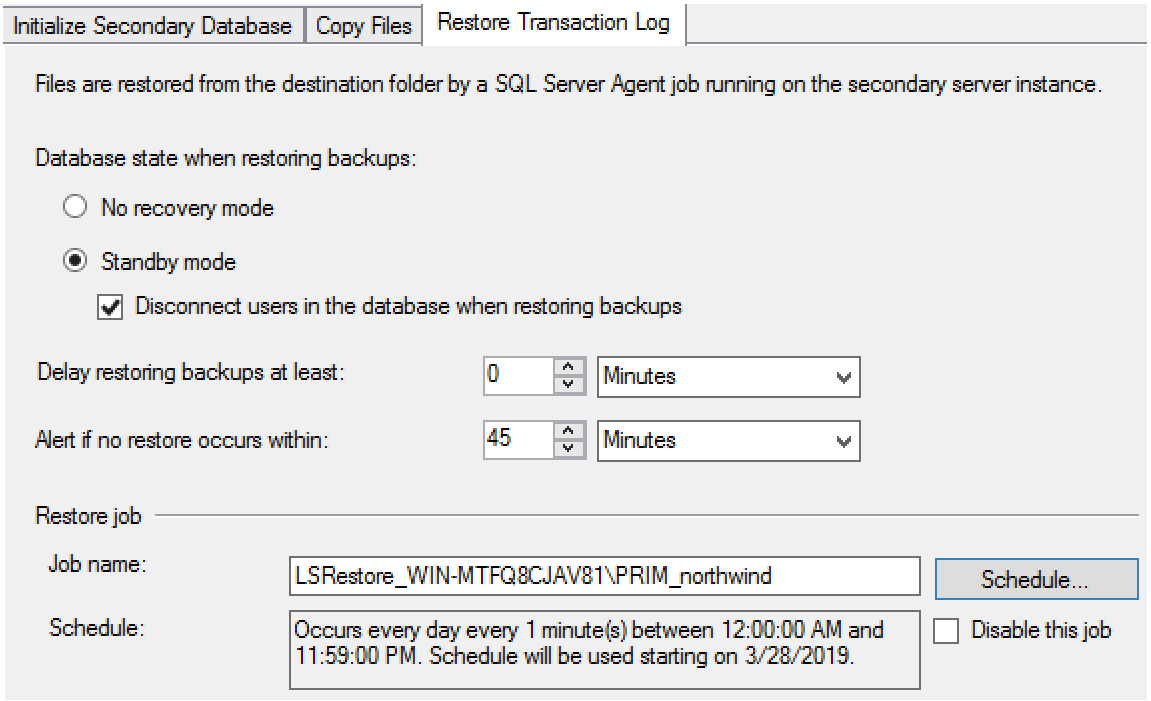
Copy job

Job name: LSCopy_WIN-MTFQ8CJAV81\PRIM_northwind

Schedule: Occurs every day every 1 minute(s) between 12:00:00 AM and 11:59:00 PM. Schedule will be used

□ Disable this job

8. On the Restore tab, select Standby mode and Disconnect users and also specify a restore schedule of every 1 minute. *Note: this setting is only for demo purposes*.



Initialize Secondary Database | Copy Files | Restore Transaction Log

Files are restored from the destination folder by a SQL Server Agent job running on the secondary server instance.

Database state when restoring backups:

○ No recovery mode

● Standby mode

☑ Disconnect users in the database when restoring backups

Delay restoring backups at least: 0 Minutes

Alert if no restore occurs within: 45 Minutes

Restore job

Job name: LSRestore_WIN-MTFQ8CJAV81\PRIM_northwind

Schedule...

Schedule: Occurs every day every 1 minute(s) between 12:00:00 AM and 11:59:00 PM. Schedule will be used starting on 3/28/2019.

□ Disable this job

9. Optionally, you can configure the THIRD server as a Monitor server. The Monitor server is the instance which monitors both the primary and secondary servers and where the log shipping alert job runs. This job generates an error if either of the three processes (backup, copy, restore) fails. Use the sa SQL Server principal to authenticate the alert job to access the primary and secondary instances. This job will generate an alert if no restore occurs within 45 minutes. Also, two alerts will automatically be setup on the monitor instance, for the failure of the primary and the secondary instances, respectively, but with empty responses (i.e. to be configured by the administrator). A typical response is sending an email to an operator.

10. Finalize and run the configuration.

11. Verify the correct operation by connecting to both databases. You may need to wait 3 minutes to see the changes in the secondary database. You can generate a report of the current status of log shipping from the popup menu of the monitor server -> Reports -> Standard Reports -> Transaction Log Shipping Status.

12. Check the contents of the log and dest folders. You should see the transaction log backups appear every minute.

13. Disable the log shipping on the PRIM server. Since the shipped database on Secondary is in warm standby state, you must execute the following commands before you can drop it (see below). Alternatively, you can set the database to single user mode on the SSMS GUI, restore the database with recovery, and set it back to multi-user.

```
use master
alter database nw_ship set single_user with rollback immediate
restore database nw_ship with recovery
```

Failover clusters

According to a definition, “Failover clustering (FC) is a Windows Server feature that enables you to group multiple servers together into a fault-tolerant cluster to increase availability and scalability of applications and services such as … Microsoft SQL Server”[10]. A FC requires a server type operating system running on more than one server machines. This is one of the reasons why we cannot demonstrate this technology in this course.

Once the FC has been setup, SQL Server instances running on the cluster members can be configured into an Always On Availability Cluster. In such a cluster, we can define a group of databases a.k.a. ‘availability databases’ that are copied to other instances and that fail over together. Besides failover support, such an architecture can be configured for automatic read load balancing for heavy load databases as well.

[10] https://docs.microsoft.com/hu-hu/windows-server/manage/windows-admin-center/use/manage-failover-clusters
https://docs.microsoft.com/hu-hu/windows-server/failover-clustering/failover-clustering-overview

4. Data quality and master data management

Types of data at a company:

- Transactional (in OLTP databases)
- Hierarchical (taxonomies, data warehouses)
- Semi-structured (XML, json)
- Unstructured (email, pdf, blogs)
- Master data
- Metadata (data on data)

The requirement for a high Data quality (DQ) is stricter than that for data integrity (e.g. when we want to correct mistyped values).

Dimensions of data quality:

- Hard dimensions
  - Completeness—do we have all the data?
  - Accuracy—are the values accurate?
  - Consistency—do data contradict among different systems?
- Soft dimensions: as perceived by the users, e.g. trust.

If both soft and hard dimensions are evaluated low, a Master Data Management (MDM) solution should be applied. MDM requires a central data storage and *Data Governance*: the process of implementing data quality, and *Data Stewards*: people responsible for the quality of particular kinds of data, e.g. a steward may be responsible for customer data. **MDM technologies are not detailed in this course**.

As for Data quality (DQ), it can be improved in a planned or in a reactive way, the latter being applicable for a small company. The success of a DQ solution can be assessed by tracking e.g. how the number of erroneous/unknown records in a table decreases over time.

DEMO: analyze the columns of tables for completeness, consistency, functional dependency etc.:

```
use AdventureworksDW2019[11]

--check if there is a functional dependency between two columns
select SalesReasonKey, SalesReasonName, SalesReasonReasonType from DimSalesReason

--the data suggests that the SalesReasonReasonType column depends on the SalesReasonName
column i.e. we can always tell the former from the latter
--check if the data supports this assumption

select SalesReasonReasonType, count(*) from DimSalesReason group by SalesReasonReasonType
--3 groups with 4,5,1 members
select SalesReasonName, count(*) from DimSalesReason group by SalesReasonName  --each
group has 1 member -> candidate key
--conclusion: the SalesReasonName is likely a subcategory of the SalesReasonReasonType
```

[11] You can download this database from https://github.com/Microsoft/sql-server-samples/releases/download/adventureworks/AdventureWorksDW2019.bak

```
--checking functional dependency
--how to check if CountryRegionCode is functionally dependent on StateProvinceCode
use AdventureworksDW2016CTP3
select * from DimGeography
select StateProvinceCode, count(*) from DimGeography group by StateProvinceCode --71
select StateProvinceCode, count(*) from DimGeography group by
StateProvinceCode,CountryRegionCode  --71
--71=71 so a StateProvinceCode will always have the same CountryRegionCode
--it is highly likely that we have a functional dependency between the two fields
select color, count(*) from dimproduct group by color --10
select ProductLine, count(*) from dimproduct group by productline --5
select ProductLine, count(*) from dimproduct group by productline, color --27 > 5, 27 >
10
--there is no functional dependency between the two fields
```

PRACTICE: analyze the columns of the DimCustomer and DimEmployee tables for completeness and functional dependency. DimCustomer: Education<->Occupation, DimEmployee: SalesTerritory<->Gender

A de facto functional dependency may be due to an unmodelled, but existing business rule of the business domain. Such a hidden relation may violate the 3NF structure via a transitive dependency within a table and it can lead to **redundancy and inconsistency**. Therefore it is the interest of the company to detect such relations by data analysis.

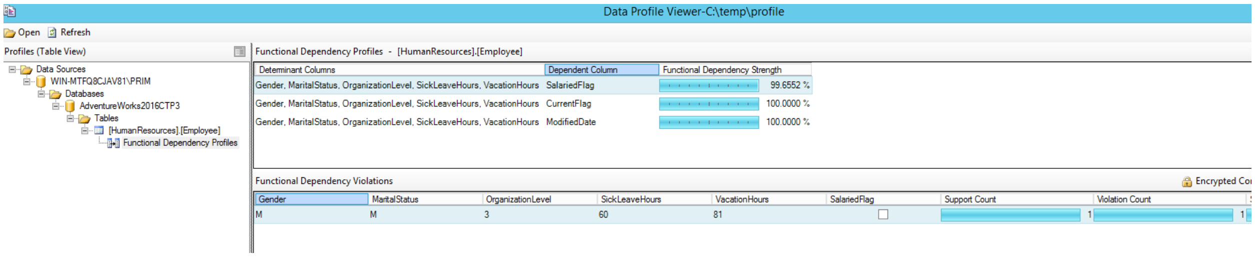
EXAMPLE: our employees have a job position code (like researcher, manager, technical or secretary) and an education level code (like elementary school, high school, Phd). There may exist a policy within the company that allows only a single education level for a certain job position, thus establishing a de facto functional dependency between the two fields.

Data profiling

Data Profiling or exploration means to automatically find statistics and data quality related features of a column:

- Identify functional dependencies, candidate keys and potential foreign keys (see above)
- Calculate column value distributions and other statistics
- Measure null ratio and length distribution for string types
- Derive *de facto* column patterns as regular expressions from the values that can be used as *domain rules* (see later)

In MS SQL Server, data profiling can be accomplished as an Integration Services Data Profiling Task that saves the results in an XML file (not detailed in this text)[12].


Data Profile Viewer-C:\temp\profile
Open
Refresh
Profiles (Table View)
Data Sources
WIN-MTFQ8CJAV81\PRIM
Databases
AdventureWorks2016CTP3
Tables
[HumanResources].[Employee]
Functional Dependency Profiles
Functional Dependency Profiles - [HumanResources].[Employee]
Functional Dependency Violations
Encrypted Co

Determinant Columns	Dependent Column	Functional Dependency Strength
Gender, MaritalStatus, OrganizationLevel, SickLeaveHours, VacationHours	SalariedFlag	99.6552 %
Gender, MaritalStatus, OrganizationLevel, SickLeaveHours, VacationHours	CurrentFlag	100.0000 %
Gender, MaritalStatus, OrganizationLevel, SickLeaveHours, VacationHours	ModifiedDate	100.0000 %

Gender	MaritalStatus	OrganizationLevel	SickLeaveHours	VacationHours	SalariedFlag	Support Count	Violation Count
M	M	3	60	81	□	1	1

[12] https://docs.microsoft.com/en-us/sql/integration-services/control-flow/data-profiling-task-and-viewer

SQL Server Data Quality Services

The DQS Service is responsible for cleaning, profiling, and matching reference data. The DQS Server includes the DQS engine and the DQS databases:

- DQS_MAIN: stored procedures that implement the DQS engine, plus Knowledge Bases, e.g. US – Last Name knowledge base, reference data: e.g. cities in Hungary. Roles in the DQS_MAIN database: dqs_admininstrator, dqs_kb_editor, dqs_kb_operator (can be assigned to different users)
- DQS_PROJECTS: data needed for cleaning and matching projects
- DQS_STAGING_DATA: temporary storage for data to be cleaned, results of cleaning

Tasks to be performed by a Data Quality client: manage knowledge bases, execute cleaning projects, administer DQS. Client implementations:

- An SQL Server Integration Services 'DQS Cleansing transformation' node can perform cleaning in the data flow of an SSIS package
- SQL Server 2017 Data Quality Client (DQC), a desktop app

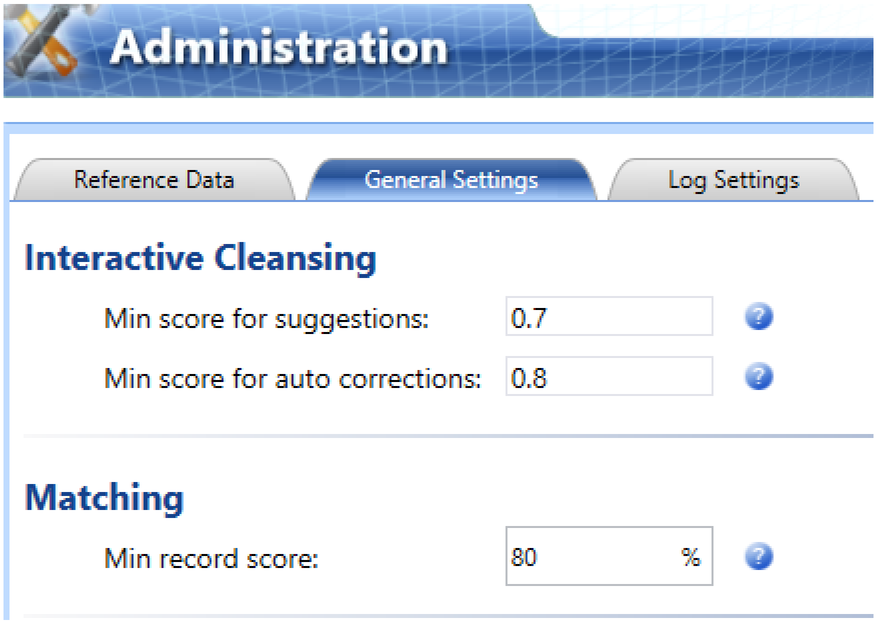
Data cleansing projects

There are two kinds of DQS projects:

- **Basic data cleansing**: the goal is to correct individual field values. We are going to demonstrate this type of project in this course.
- **Identity mapping and de-duplicating** according to matching rules: the goal is to identify and remove possibly duplicated records i.e. entities. The matching rule is defined at the KB level using common distance measures of data science. We are not going to demonstrate this type of project in this course notes.

Both types are based on a knowledge base (KB) already installed in the DQS_MAIN database, and use these parameters:

- Cleansing: the Min Score for Suggestions parameter sets the minimal similarity for generating a suggestion. It must be *less* than the Min Score for Auto Correction parameter.
- Matching (de-duplication): threshold value for the matching policy. It is a measure of record similarity.



Interactive Cleansing

Min score for suggestions: 0.7

Min score for auto corrections: 0.8

Matching

Min record score: 80 %

A DQS Knowledge Base (KB) may contain several **domains** i.e. reference value sets used for cleansing. A domain is defined by the following components:

- **Name** and **data type** e.g. string
- Settings for normalization like converting to uppercase or removing dummy characters, formatting and spell checking
- Reference data, which may be stored in the DQS_MAIN database or alternatively, may be provided by an external service
    - The reference values, called **domain values**. A domain value is a tuple that contains a **leading value** and a list of its synonyms
- **Domain rules**, similar to CHECK constraints
- **Term based relations** that apply only to a part of a value, e.g. ‘%Ltd.%’ -> ‘%Limited Company%’

Composite domains can be created of semantically connected fields e.g. city, street, etc. parts of an address stored in more than one columns.

We have some default KBs already installed in the DQS_MAIN on the PRIM server like Country/Region, US-Counties, US-Last names, etc. We can also develop our own KB’s.

DEMO: We clean the last names of a table using the default KB

Start the DQC and type the server name: WIN-MTFQ8CJAV81\PRIM.

- On the Administration->Configuration->General settings panel, review the Min Score for Suggestions and Min Score for Auto Correction settings. These settings may be adjusted on a trial and error basis
- On the Knowledge base management panel, review the pre-installed KBs
- Create a new data cleansing project called Lastnames.
- Select the KB: DQS Data and the US_Last Name domain. The activity should be Cleansing

- On the Map page, select the AdventureworksDW2016CTP3 database, the DimCustomer table and the Lastname field and map it to the US_Last Name domain
- Start the cleaning. Since we have more than 18000 last names in the DimCustomer table to cleanse, pre-processing will run for ca. 5 minutes, cleansing for ca. 30 s[13]
- Check the hard dimensions of Completeness (100%) and Accuracy (98%). After cleansing, a value may be in one of these states:
  - **Correct**: found in the domain values
  - **Corrected**: changed automatically to a domain value because the similarity was above the Min Score for Auto Correction
  - **Suggested**: a new value is only suggested because the similarity was above the Min Score for Suggestions
  - **New**: a similar value is not found among the domain values but the value conforms to the domain rules
  - **Invalid**: a new value that contradicts the domain rules
- With the above Min Score settings, the process finds some cases where a correction is suggested and 3 cases where a correction was performed automatically. 328 names were correctly found in the KB.
- Try to Approve a suggested change -> the row will be relocated onto the Corrected pane
- Finally, save the cleansed table containing the LastNameSource and LastNameOutput columns in the original database and process it with T-SQL commands.

PRACTICE: cleanse the AdventureworksDW2016CTP3.DimGeography.EnglishCountryRegionName field using the Country/Region domain. Try again after adding a misspelled country to the table (Austrila).

Building your own Knowledge Base

We can use an existing set of values in a table field to construct our custom KB. The steps:

1. Automatic knowledge discovery using the data sample
2. Manual domain management and configuration

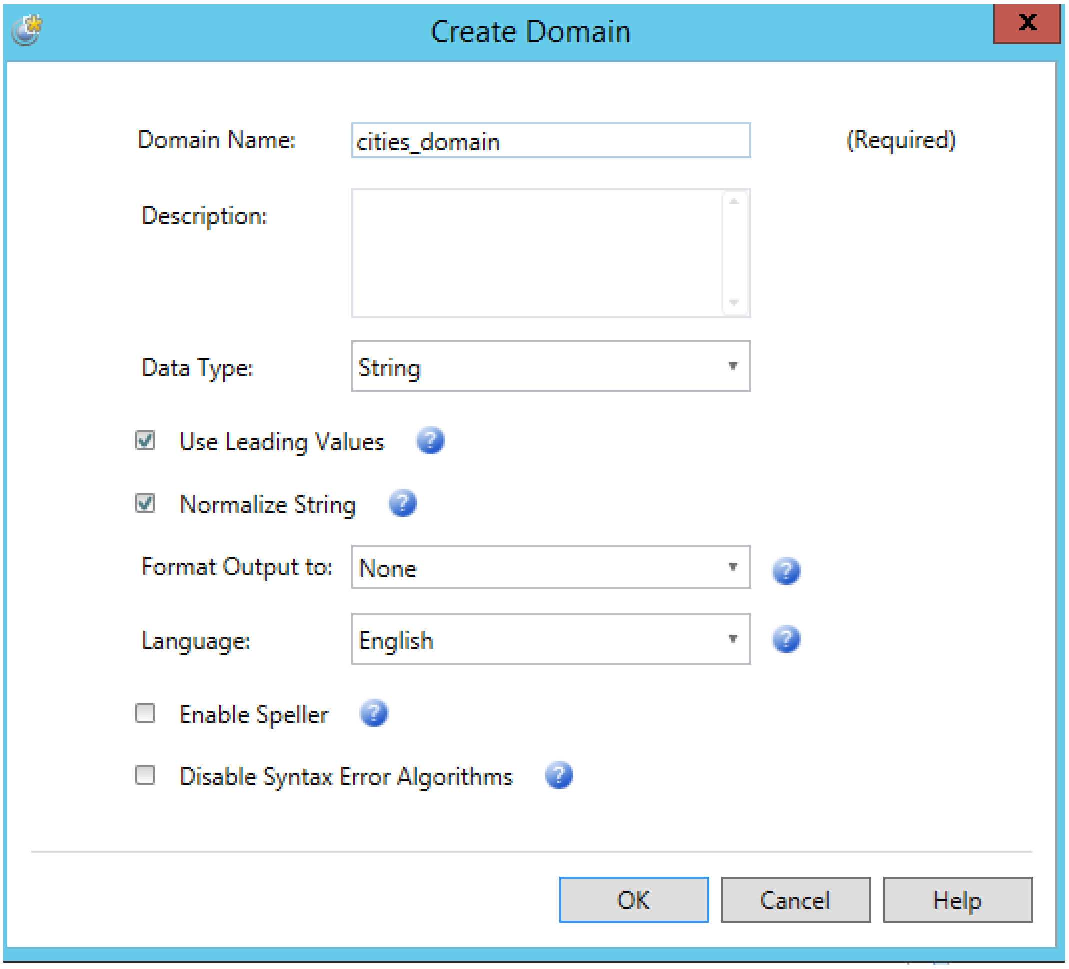
DEMO: we create a KB

- Create a new view in the DQS_STAGING_DATA database

```
use DQS_STAGING_DATA
go
create view vi_places as
select distinct City, StateProvinceName StateProvince, EnglishCountryRegionName
CountryRegion
from AdventureworksDW2016CTP3.dbo.DimGeography
go
```

[13] If you receive a ‘Parallel processing task failed’ error during pre-processing, check the DQServerLog.DQS_MAIN.log file in the Log directory of the instance. The error is *‘The DELETE statement conflicted with the REFERENCE constraint "B_INDEX_LEXICON_EXTENSION_B_INDEX_LEXICON_FK". The conflict occurred in database "DQS_PROJECTS", table "DQProject1000001.B_INDEX_LEXICON_EXTENSION", column 'TERM_ID'’*. Using the SSMS Modify keys panel, set the On Delete action of the B_INDEX_LEXICON_EXTENSION_B_INDEX_LEXICON_FK foreign key constraint to CASCADE.

- Start the DQS client and create a new KB called places
- Select Knowledge discovery, set the source to the vi_places and create two domains called cities_domain and countryregion_domain using the fields of the same name of the vi_places view



Create Domain

Domain Name: cities_domain (Required)

Description:

Data Type: String

☑ Use Leading Values

☑ Normalize String

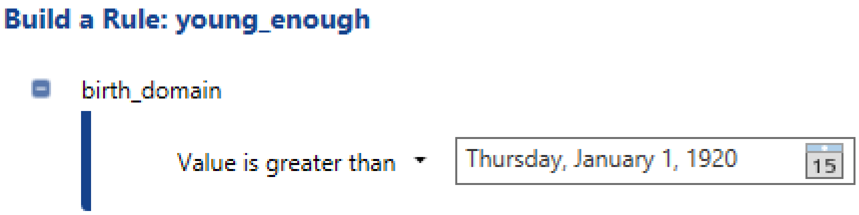
Format Output to: None

Language: English

☐ Enable Speller

☐ Disable Syntax Error Algorithms

- Start the discovery
- In the slot of Muehlheim set the type to **Error** and manually enter Mühlheim in the ‘Correct to’ field, *then press Enter*
- Select the new KB from the main menu, and from its popup menu select Domain management
- Manually add a new domain called birth_domain with a simple domain rule:



Build a Rule: young_enough

birth_domain

Value is greater than Thursday, January 1, 1920

- Publish the KB

- Select the new KB for management and export it to a file called places.dqs
- Create a new table test_customer in the DQS_STAGING_DATA database from the Dimcustomer table joined to the Dimgeography table on the GeographyKey column
- Change the Birthdate of the first customer to 1925-01-02 and the city to Muehlheim. Change the city of the second customer to Piripócs.
- Use the new 'places' KB to cleanse the test_customer table and review the results

PRACTICE: Build a KB using the DimProduct table, ProductName field and demonstrate its application as above.

There are also free, independent tools available for data cleaning and master data management e.g. https://openrefine.org/

5. Columnstore and partitioning

Columnstore

The columnstore is an alternative way of storing relational data. Recommended for l**arge data warehouse tables with many fields** and **infrequent changes**. In the table blocks (pages), the fields for a record are not stored together, rather, each field is stored separately, and the values for a record are linked together[14]. It owes its speed to the few I/O operations (block reads) due to the compression of the pages[15] (reduced size -> fewer pages) and the fact that queries usually do not require reading the whole record, only a small number of fields. This is particularly apparent when the table has a very large number of fields, which is typical for data warehouses. Compression types:

- Row compression substitutes fixed length data types with variable length data, e.g. a 4-byte integer field can be stored on only 1 byte depending on the actual values. Applicable to OLTP applications.
- Page compression compresses whole pages and it is more applicable to data warehouse applications or to OLTP applications with infrequent updates. Compression is particularly effective when the values of a field are quite similar a.k.a. *low cardinality* fields.

The free edition of Postgres currently does not support compression.

```
use AdventureWorksDW2016_ext

/*
--create a BTREE table
select * into FactResellerSalesXL_BTREE from FactResellerSalesXL_CCI
alter table FactResellerSalesXL_BTREE alter column SalesOrderLineNumber tinyint not null
alter table FactResellerSalesXL_BTREE alter column SalesOrderNumber nvarchar(20) not null
alter table FactResellerSalesXL_BTREE add constraint c1 primary key
(SalesOrderLineNumber, SalesOrderNumber)
*/

--RESTART SERVER
set statistics time on
set statistics io on
go
select count(*) from FactResellerSalesXL_BTREE  --11669638
exec sp_spaceused 'dbo.FactResellerSalesXL_BTREE', @updateusage = 'TRUE'
--data: 2523168 KB KB, index: 9776 KB KB

select count(*) from FactResellerSalesXL_PageCompressed  --11669638
exec sp_spaceused 'dbo.FactResellerSalesXL_PageCompressed', @updateusage = 'TRUE'
--data: 695624 KB KB KB, index: 2344 KB KB KB

select count(*) from FactResellerSalesXL_CCI  --11669638
exec sp_spaceused 'dbo.FactResellerSalesXL_CCI', @updateusage = 'TRUE'
--data: 525344 KB, index: 157624 KB
```

[14] https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16

[15] https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression?view=sql-server-ver16

```
go
dbcc freeproccache --
dbcc dropcleanbuffers -- empty data buffer

--B-TREE
--=======
select b.SalesTerritoryRegion
    ,FirstName + ' ' + LastName as FullName
    ,count(SalesOrderNumber) as NumSales
    ,sum(SalesAmount) as TotalSalesAmt
    ,Avg(SalesAmount) as AvgSalesAmt
    ,count(distinct SalesOrderNumber) as NumOrders
    ,count(distinct ResellerKey) as NumResellers
from FactResellerSalesXL_BTREE a
inner join DimSalesTerritory b on b.SalesTerritoryKey = a.SalesTerritoryKey
inner join DimEmployee d on d.Employeekey = a.EmployeeKey
inner join DimDate c on c.DateKey = a.OrderDateKey
where b.SalesTerritoryKey = 3 and c.FullDateAlternateKey between '1/1/2006' and
'1/1/2010'
group by b.SalesTerritoryRegion,d.EmployeeKey,d.FirstName,d.LastName,c.CalendarYear
 --CPU time = 3949 ms, elapsed time = 9568 ms

--DATA_COMPRESSION = PAGE
--=================================
go
select b.SalesTerritoryRegion
    ,FirstName + ' ' + LastName as FullName
    ,count(SalesOrderNumber) as NumSales
    ,sum(SalesAmount) as TotalSalesAmt
    ,Avg(SalesAmount) as AvgSalesAmt
    ,count(distinct SalesOrderNumber) as NumOrders
    ,count(distinct ResellerKey) as NumResellers
from FactResellerSalesXL_PageCompressed a
inner join DimSalesTerritory b on b.SalesTerritoryKey = a.SalesTerritoryKey
inner join DimEmployee d on d.Employeekey = a.EmployeeKey
inner join DimDate c on c.DateKey = a.OrderDateKey
where b.SalesTerritoryKey = 3
    and c.FullDateAlternateKey between '1/1/2006' and '1/1/2010'
group by b.SalesTerritoryRegion,d.EmployeeKey,d.FirstName,d.LastName,c.CalendarYear
GO  -- CPU time = 3264 ms,  elapsed time = 3776 ms.

--column store with page compression
--create clustered columnstore index [IndFactResellerSalesXL_CCI] on
[dbo].[FactResellerSalesXL_CCI]
--with (drop_existing = off, compression_delay = 0, data_compression = columnstore) on
[primary]
--================================
select b.SalesTerritoryRegion
    ,FirstName + ' ' + LastName as FullName
    ,count(SalesOrderNumber) as NumSales
    ,sum(SalesAmount) as TotalSalesAmt
    ,Avg(SalesAmount) as AvgSalesAmt
    ,count(distinct SalesOrderNumber) as NumOrders
    ,count(distinct ResellerKey) as NumResellers
from FactResellerSalesXL_CCI a
inner join DimSalesTerritory b on b.SalesTerritoryKey = a.SalesTerritoryKey
inner join DimEmployee d on d.Employeekey = a.EmployeeKey
inner join DimDate c on c.DateKey = a.OrderDateKey
where b.SalesTerritoryKey = 3
    and c.FullDateAlternateKey between '1/1/2006' and '1/1/2010'
group by b.SalesTerritoryRegion,d.EmployeeKey,d.FirstName,d.LastName,c.CalendarYear
GO  -- CPU time = 360 ms,  elapsed time = 492 ms
```

A note on the reads statistics:

- **Logical reads** are 8KB page reads from the data buffer
- **Physical reads** are reads that had to be fetched from the storage (from the OS), these are blocking the execution of the query
- **Read-ahead reads** are asynch read requests to the OS for pages that are likely to be needed by the query. These do not block the query.

Partitioning

The purpose of partitioning: divide and conquer. When a table is partitioned along a field:

- Since the physical reads are the bottleneck of database query execution, queries run faster if each partition is on a different physical storage, because the reads can be parallelized. For most queries, reading only a few partitions may be enough (not all of them).
- Large tables often have limited load times, e.g. nightly in a banking system etc., for times when the data does not change. INSERT and DELETE are logged statements (implicit, logged transactions), so inserting or deleting too many records at once is slow (e.g. indexes have to be rebuilt afterwards). Partitioning makes loading faster, because instead of a large table insert, we write to an empty table partition, which is a fast 'minimally logged' operation (in simple recovery mode).

It is hard to determine the exact table size where partitioning is already worthwhile, but a rule of thumb is that the size of the table should exceed the physical memory of the database server.

A typical partitioning key is the date, e.g. by month or year. The table's columnstore index, if it has one, should also be partitioned (aligned index). Then partition switching does not require rebuilding the index, only metadata is changed. Essential partitioning concepts on SQL Server:

- **Partitioning scheme**: assigns partitions to groups of files
- **Partitioning function**: assigns each record to a partition based on the underlying field(s) of the partitioning, such as the date

Note: The size (number of records) of partitions can be different.

A recommended best practice for partitioning and loading a large table with minimal logging (e.g. when new data arrives into the data warehouse) is as follows.

1. Create the partitioning scheme and function
2. Load existing data
3. In order to load the new data, create an empty auxiliary (staging) table with the same schema (and compression) as the table

4. In this table, a check constraint protects the partitioning key so that no wrong data can be inserted
5. The new data is loaded into the auxiliary table and a columnstore index is created
6. Swap the auxiliary table to the next empty partition of the fact table
7. Before loading the next partition, we must delete the columnstore index and update the check constraint accordingly

```
set statistics time off
set statistics io off
--by year:
select min(OrderDate), max(OrderDate)  from FactInternetSales s   --2010..2014
--we'll have 5 partitions for the 5 years
--drop table InternetSales
go
--part. function:
--drop partition function PfInternetSalesYear
create partition function PfInternetSalesYear (tinyint) as range left for values (10, 11,
12, 13)
--e.g. '10' will mean that the year <= 2010 (due to the LEFT)
--TINYINT: 1 byte, 0..255
--part. scheme: all in the same filegroup
--drop partition scheme PsInternetSalesYear
create partition scheme PsInternetSalesYear as partition PfInternetSalesYear all to
([PRIMARY])
go
--Note: we must combine the identity key with the part. number because the identity may
be unique only within the part.
--drop table InternetSales
create table InternetSales(
       InternetSalesKey int not null identity(1,1),
       PcInternetSalesYear tinyint not null,  --part. number
       ProductKey int not null,
       DateKey int not null,
       OrderQuantity smallint not null default 0,
       SalesAmount money not null default 0,
       UnitPrice money not null default 0,
       DiscountAmount float not null default 0,
       constraint PK_InternetSales primary key (InternetSalesKey, PcInternetSalesYear)
)
ON PsInternetSalesYear(PcInternetSalesYear) --this'll make the partitions
GO
--adding external keys and page compression
--ALTER TABLE dbo.InternetSales ADD CONSTRAINT FK_InternetSales_Customers FOREIGN
KEY(CustomerDwKey)
--REFERENCES dbo.Customers (CustomerDwKey);
ALTER TABLE dbo.InternetSales ADD CONSTRAINT FK_InternetSales_Products FOREIGN
KEY(ProductKey)
REFERENCES dbo.DimProduct (ProductKey);
ALTER TABLE dbo.InternetSales ADD CONSTRAINT FK_InternetSales_Dates FOREIGN KEY(DateKey)
REFERENCES dbo.DimDate (DateKey);
ALTER TABLE dbo.InternetSales REBUILD WITH (DATA_COMPRESSION = PAGE);
GO
--load data up to 2013
INSERT INTO dbo.InternetSales (PcInternetSalesYear, ProductKey, DateKey,
OrderQuantity, SalesAmount, UnitPrice, DiscountAmount)
```

```
SELECT CAST(SUBSTRING(CAST(OrderDateKey AS CHAR(8)), 3, 2) AS TINYINT) AS
PcInternetSalesYear,
--note the trick: dates are stored as int e.g. 20110223 to save space
       ProductKey, OrderDateKey, OrderQuantity, SalesAmount, UnitPrice, DiscountAmount
FROM FactInternetSales AS FIS
WHERE CAST(SUBSTRING(CAST(FIS.OrderDateKey AS CHAR(8)), 3, 2) AS TINYINT) < 13
GO
--how many records in partitions?
SELECT $PARTITION.PfInternetSalesYear(PcInternetSalesYear) AS PartitionNumber, COUNT(*)
AS NumberOfRows
FROM InternetSales GROUP BY $PARTITION.PfInternetSalesYear(PcInternetSalesYear)
PartitionNumber      NumberOfRows
1                           14           --2010
2                           2216   --2011
3                           3397   --2012
--the last partition is still empty

--columnstore:
CREATE COLUMNSTORE INDEX CSI_InternetSales ON dbo.InternetSales
       (InternetSalesKey, PcInternetSalesYear, ProductKey, DateKey,
       OrderQuantity, SalesAmount, UnitPrice, DiscountAmount)
ON PsInternetSalesYear(PcInternetSalesYear) --Note: the index is aligned to the partition
GO

--a new staging table to receive the year 2013 data
--in order to avoid errors we use a check constraint
create TABLE dbo.InternetSalesNew (
       InternetSalesKey INT NOT NULL IDENTITY(1,1),
       PcInternetSalesYear TINYINT NOT NULL CHECK (PcInternetSalesYear = 13), --check
       ProductKey INT NOT NULL,
       DateKey INT NOT NULL,
       OrderQuantity SMALLINT NOT NULL DEFAULT 0,
       SalesAmount MONEY NOT NULL DEFAULT 0,
       UnitPrice MONEY NOT NULL DEFAULT 0,
       DiscountAmount FLOAT NOT NULL DEFAULT 0,
       CONSTRAINT PK_InternetSalesNew PRIMARY KEY (InternetSalesKey, PcInternetSalesYear)
)
GO
ALTER TABLE dbo.InternetSalesNew ADD CONSTRAINT FK_InternetSalesNew_Products FOREIGN
KEY(ProductKey) REFERENCES dbo.dimProduct (ProductKey);
ALTER TABLE dbo.InternetSalesNew ADD CONSTRAINT FK_InternetSalesNew_Dates FOREIGN
KEY(DateKey) REFERENCES dbo.dimDate (DateKey);
go
ALTER TABLE dbo.InternetSalesNew REBUILD WITH (DATA_COMPRESSION = PAGE)
GO
--load 2013
--since the table was empty, the insert will not be logged
INSERT INTO dbo.InternetSalesNew (PcInternetSalesYear,ProductKey, DateKey,
       OrderQuantity, SalesAmount,UnitPrice, DiscountAmount)
SELECT CAST(SUBSTRING(CAST(OrderDateKey AS CHAR(8)), 3, 2) AS TINYINT) AS
PcInternetSalesYear,
       ProductKey, OrderDateKey,OrderQuantity, SalesAmount, UnitPrice, DiscountAmount
FROM FactInternetSales
WHERE CAST(SUBSTRING(CAST(OrderDateKey AS CHAR(8)), 3, 2) AS TINYINT) = 13
GO
--columnstore for the staging table
CREATE COLUMNSTORE INDEX CSI_InternetSalesNew
ON dbo.InternetSalesNew (InternetSalesKey, PcInternetSalesYear, ProductKey, DateKey,
OrderQuantity, SalesAmount, UnitPrice, DiscountAmount);
GO
--the key step is loading the partition 4
ALTER TABLE dbo.InternetSalesNew SWITCH TO dbo.InternetSales PARTITION 4
--this required no data transfer
--partitions:
SELECT $PARTITION.PfInternetSalesYear(PcInternetSalesYear) AS PartitionNumber, COUNT(*)
AS NumberOfRows
FROM dbo.InternetSales GROUP BY $PARTITION.PfInternetSalesYear(PcInternetSalesYear)
order by PartitionNumber
PartitionNumber      NumberOfRows
1                           14
2                           2216
3                           3397
4                           52801
select count(*) from InternetSalesNew --0!
```

You can do all this also in the Storage menu of the SSMS GUI.

The Postgres alternative needs the direct manipulation of the partitions as tables.

```
select * from products where productname <'N'
select * from products where productname >='N'

--drop table products_p
create table products_p --the parent table is virtual, does not store data
       (productid int, productname varchar(40), part_key char(1), unitprice money)
partition by range(part_key)

create table products_p_a_m partition of products_p for values from ('a') to ('m');  --
mind the overlap
create table products_p_m_z partition of products_p for values from ('m') to ('z');
--create table products_p_n_z partition of products_p for values from ('n') to ('z')
tablespace very_fast_drive;

insert into products_p (productid, productname, part_key, unitprice)
       select productid, productname, substring(productname, 1,1), unitprice from
products
--queries are redirected to the right partition
select * from products_p where productname <'d' --5
--the partitions can be queried also directly
select * from products_p_a_m where productname <'d' --5
select * from products_p_m_z where productname <'d' --0

--partitons can be managed independently
drop table products_p_a_m;
--remove from the parent table but keep the data
alter table products_p detach partition products_p_a_m;
--for loading, the new partiton can be created independently, then attached to the parent
table
alter table products_p attach partition products_p_a_m for values from ('a') to ('m');
--it is recommended to create check constraints on partitions -> helps the query
optimizer
```

In Postgres, you can also use the table inheritance mechanism to support partitioning.

PRACTICE: create a partitioning scheme for the FactResellerSales table according to the DueDateKey field and load it partition by partition.

6. Cloud database technologies

Overview

Cloud providers such as Amazon, Google, IBM, Microsoft offer database management on a SaaS or PaaS basis. The viability of such a solution instead of running private database servers depends on economical and, to some extent, technical considerations. Some of these:

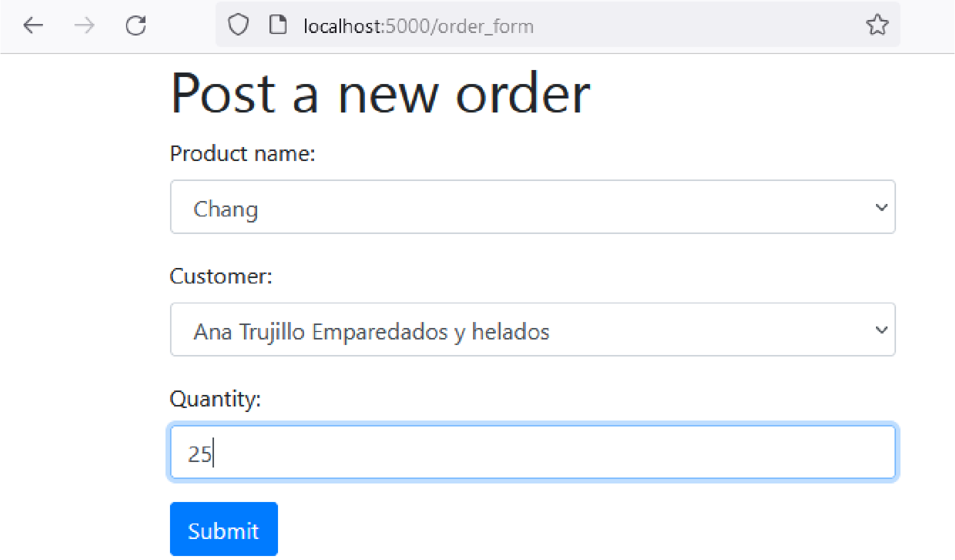
- ✓ Cloud services offer high availability (better than the corporate servers) with a geographically distributed infrastructure.
- ✓ Cloud services offer automated on-demand scalability with flexible, at least in part load-dependent, billing schemes.
- ✓ The cloud offers several other in-cloud services like data analysis, image processing, NLP, etc. which can access the cloud database and which can be integrated in a solution.
- ✗ The functionality offered by the cloud database services is typically inferior to the private servers, though some functions may be replaced by other means.
- ✗ Though some costs are proportional with the usage, there are also fixed costs to be paid for e.g. data storage. All such costs should be considered in the business plan of the planned service or solution.
- ✗ Data security may be compromised if highly sensitive data (like personal healthcare records) physically leave the company network.
- ✗ Critical services, like production system control should not depend on Internet access. These latter two concerns can be alleviated in a **private cloud**[16].

A simple database app in the GCP cloud

Our demo app

- Selecting a product from a dropdown list, specifying a quantity and posting an order in the Northwind database. The returned page confirms the order and lists the last 5 orders.

[16] E.g. https://www.openstack.org/



localhost:5000/order_form

Post a new order

Product name:

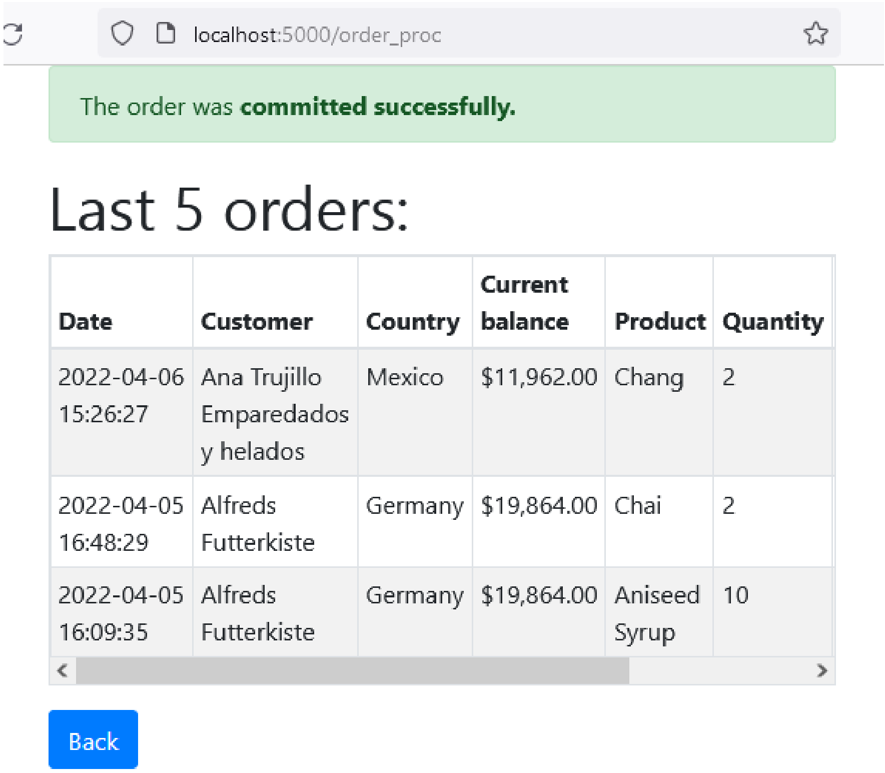
Chang

Customer:

Ana Trujillo Emparedados y helados

Quantity:

25



localhost:5000/order_proc

The order was **committed successfully.**

Last 5 orders:

Date	Customer	Country	Current balance	Product	Quantity
2022-04-06 15:26:27	Ana Trujillo Emparedados y helados	Mexico	$11,962.00	Chang	2
2022-04-05 16:48:29	Alfreds Futterkiste	Germany	$19,864.00	Chai	2
2022-04-05 16:09:35	Alfreds Futterkiste	Germany	$19,864.00	Aniseed Syrup	10

Postgres backend

Relational->relational migration can be performed by import/export functions or by manually editing a dump created from the source database, but in several cases only a specialized tool or custom dedicated software can solve the problem. Typical issues:

- Data types
- Differences in SQL DDL/DML syntax
- Differences in the authentication models

- Any server side logic, like stored procedures, must be re-implemented

In this demo, we migrate the SQL Server Northwind database to Postgres. Install Postgres and pgAdmin, then run the pg_script.sql script in the database postgres. The contents of the script is as follows.

1. Creates the Products, Orders, Order details, Customers tables and inserts data
2. Creates a new stored function that supports the creation of a new order
3. Creates a new view that lists the last 5 orders' properties

Notes:

The script was manually edited based on a dump script generated by SSMS. More on this problem: https://severalnines.com/database-blog/migrating-mssql-postgresql-what-you-should-know

An example for two dialects of a default constraint:

- Postgres: `ALTER TABLE products ALTER COLUMN Unitprice SET DEFAULT 0`
- SQL Server: `ALTER TABLE products ADD CONSTRAINT DF_Products_UnitPrice  DEFAULT ((0)) FOR UnitPrice`

Another example for the Boolean type: SQL Server accepts a 0/1, while Postgres does not.

The script contains a stored function that implements a transaction. The **atomicity** of this transaction is an important business rule that must be observed.

A simple Python flask app

Since the GCP console was designed for the management of the cloud infrastructure, and not for app development, first we develop the app on localhost:

1. (*already done on the VM*) Install Anaconda python:
   https://www.anaconda.com/products/individual
2. (*already done on the VM*) Create a virtual environment for the project[17]:
   a. Anaconda admin prompt: `set https_proxy=http://proxy.uni-pannon.hu:3128`
      (otherwise you'll receive 'SSL: WRONG_VERSION_NUMBER')
   b. `conda create -n flask_demo pip` //python=3.8.5
      (pip is added as a dependency)
      python version: `python -version` //3.9.11
      available virtual environments: `conda env list`)
   c. `conda activate flask_demo` (later we'll install the necessary packages into the flask_demo environment)
   d. `pip install flask`
   e. `pip install psycopg2`
3. download the northwind.py app, and cd to the folder of your flask project (i.e. the directory named 'flask')

[17] https://blog.usejournal.com/why-and-how-to-make-a-requirements-txt-f329c685181e

4. start the Anaconda Spyder python editor and set the password to … and the port to 5432 in line 59. Review the app. Note that the 'with connection:' block runs the calling of the new_order() function in a **transaction**.
5. At the conda prompt: `set FLASK_APP=northwind.py` (by setting set DEBUG=1 you won't need to manually restart the web server every time). Note: we'll use main.py as app name on the GCP so that no specific setting should be necessary.
6. `flask run`
7. Test the page at http://localhost:5000/order_form Technologies used:
    a. The templates folder holds the html pages which are processed by the jinja template engine. The python variables are available in the HTML templates {% for ... in / if …else / with / block, extends %}, and the templates can be bested with the 'block'[18].
    b. Bootstrap (formatting)
8. Could be made better or more secure, but this is not our focus now[19].
9. At the conda prompt, export the virtual environment dependencies in a file: `pip freeze > requirements.txt`
10. Create an App.yaml file in the project directory with this content: `runtime: python39` More on Yaml: https://cloud.google.com/appengine/docs/standard/python3/config/appref

Start a new GCP project

In this project we'll use the App engine to run our Flask app, a cloud SQL database and a Firestore document store.

1. Start a private browser and enable pop-up windows
2. Activate your $50 coupon using your university email address, observing the instructions you received in the Neptun message
3. Log in to https://console.cloud.google.com
4. Start a new project or delete existing: IAM & Admin -> Manage resources
5. You must specify a unique project name, which will be referred to as YOUR-PROJECT-ID in this manual. Check the billing account linked to the project. It should have $50 credit at startup.
6. On the Firestore page set: SELECT NATIVE MODE, then CREATE DATABASE. Only a single Firestore database is allowed in a GCP project.

Migrate the Postgres database to GCP

1. GCP Databases->SQL->Create instance -> PostgreSQL (you must enable the Compute Engine API)
2. Choose an instance id and password. This is the name of the database server instance, not the name of the database or the database user. Set:

[18] Flask app: https://www.youtube.com/watch?v=MwZwr5Tvyxo
HTML Templates https://www.youtube.com/watch?v=QnDWIZuWYW0

[19] WTF Forms https://www.youtube.com/watch?v=UIJKdCIEXUQ
Database ORM https://www.youtube.com/watch?v=cYWiDiIUxQc

region: Europe, PostgreSQL version 13, single zone (no failover),
customize your instance: public IP enabled

3. connections: „All apps in this project are authorized by default", however, we plan to access the database from the Internet by PgAdmin, so we allow any IP address (=pg_hba.conf entry). Add network: `0.0.0.0/0`. Save.
4. In Advanced options, enable Query insights.
5. Wait while the instance is being created and then change the password of the Postgres user. Navigate to SQL Overview to find the public IP address of the server.
6. In PgAdmin connect to the cloud instance using the public IP address, then review and run the dump file pg_script.sql.
7. Change the IP in the python app to the public GCP IP and test the app. It should works just as with the local server. Check that the new orders appear in the Postgres tables.
8. Returning to GCP, open Query insights and check the execution plan of the query SELECT * FROM last_orders. Note that it may take up to 10 minutes for a query to appear in the Query insights statistics, which is an offline analysis tool.

Implementing the demo app on GCP[20] -> Cloud programming MSc course

We'll deploy the app to the GCP app engine[21].

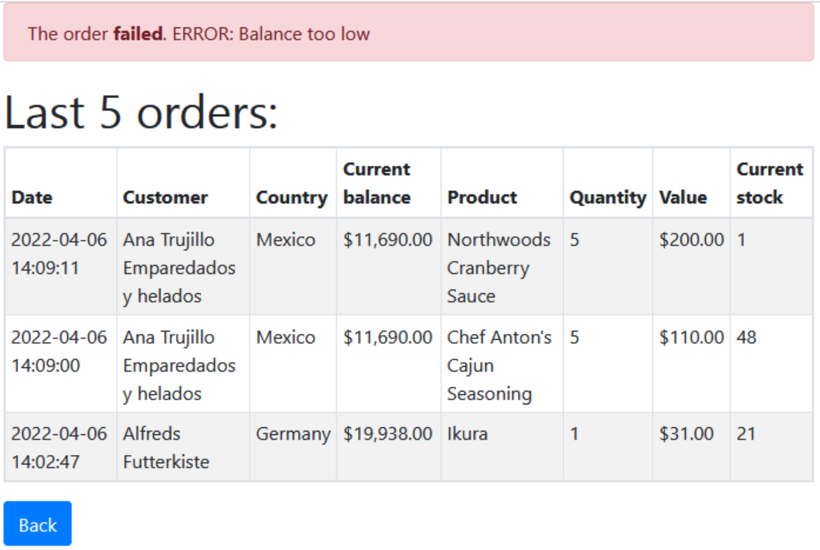
1. (*already done on the VM*) Install the Cloud SDK and the gcloud CLI on your desktop computer https://cloud.google.com/sdk/docs/install
2. Open a cmd terminal **as Windows Administrator,** cd to the flask_gcp directory, and set HTTPS_PROXY=http://proxy.uni-pannon.hu:3128
3. at the prompt type: `gcloud init` (requires authentication to our personal google profile) and select YOUR-PROJECT-ID
4. The SDK does not contain the App engine extension, so this must be installed separately. At the Google Cloud SDK shell prompt type: `gcloud components install app-engine-python` This will take some minutes.
5. `gcloud config set project YOUR-PROJECT-ID`
6. //check billing: `gcloud beta billing accounts list`
7. In the northwind.py script change the database connection parameters to the public GCP IP address, and save it as main.py. (~~Update the requirements.txt file.~~)
8. We'll use the Cloud Build API to create the container that runs the app, so this API must be enabled in the project: `gcloud services enable cloudbuild.googleapis.com`
9. Initialize the app engine in the project: `gcloud app create --project=YOUR-PROJECT-ID`
10. `gcloud app deploy`
    You'll receive target URL on which you can access the application. You can also start a browser for the app by: `gcloud app browse`

[20] If you are already familiar with cloud programming concepts, you can skip this section and go on with the app running on localhost

[21] https://medium.com/@dmahugh_70618/deploying-a-flask-app-to-google-app-engine-faa883b5ffab

11. You can review runtime error messages on the web based GCP console App engine -> Services -> Diagnose : Logs

PRACTICE: improve the GCP app by displaying also the cause of an eventual error i.e. either low balance or low stock, see below. This requires a slight modification of the new_order() stored procedure as well as the client Html templates. Deploy the modified app by typing `gcloud app deploy`



The order **failed**. ERROR: Balance too low

Last 5 orders:

Date	Customer	Country	Current balance	Product	Quantity	Value	Current stock
2022-04-06 14:09:11	Ana Trujillo Emparedados y helados	Mexico	$11,690.00	Northwoods Cranberry Sauce	5	$200.00	1
2022-04-06 14:09:00	Ana Trujillo Emparedados y helados	Mexico	$11,690.00	Chef Anton's Cajun Seasoning	5	$110.00	48
2022-04-06 14:02:47	Alfreds Futterkiste	Germany	$19,938.00	Ikura	1	$31.00	21

MORE PRACTICE: **Test the atomicity of the order transaction**. On the PgAdmin console, inject an error in the new_order() function by replacing the statement:

insert into orders (orderid, customerid) values (var_orderid, var_custid); with this statement

insert into orders (orderid, customerid) values (var_orderid, 'ERROR');

This will violate the foreign key constraint and result in a runtime error at the next invocation. Using the web app, place a new order and verify that after the error the balance of the customer is restored to the original value, even though it has already been decreased before the insert statement. *The transaction has been rolled back automatically*.

EVEN MORE PRACTICE: Split the table above in two tables that show the last 5 orders and the balance of the customers involved separately.

Before you go…

1. Stop the app: App engine -> Settings -> Disable application
2. Stop the Postgres server: SQL -> Overview -> STOP
3. Shut down the GCP project

Migration to a document store

Data models: relational vs document store

The relational data model is determined by the domain model and it is normalized in 3NF. The relational model is universal, any query can be run efficiently. The model can be tuned to a certain application via indexes.

With noSQL models like document stores and key-value stores, the proper and efficient data model can be determined only with respect to the planned application and the expected queries. For this reason, while a relational model can be migrated quite smoothly into another relational technology, there is no easy or automated solution for migrating a relational model into a noSQL model.

Migration is particularly cumbersome for relational models with complex relational structures among the tables. While hierarchical one-to-many relations map easily to noSQL structures, individual considerations are required to implement **many-to-many** relations. The structure is to be planned manually and a custom application must be implemented to manage the data loading process.

Cloud Firestore overview

Google Cloud Firestore is a document store. It supports the storage of multiple document collections. It is the new edition of the previous Realtime Database that used a single monolithic json[22]. Features of Cloud Firestore:

- „Firestore database=Collections of documents”.
- Every document is essentially a json record e.g.
  ```
  name :
          first : "Joe"
          last : "Long"
  born : 1995
  ```
- The collection **has no schema** i.e. it may store documents of arbitrary structure in the same collection. Nevertheless, even if there is no schema validation upon insertion, any realistic application reading the collection expects a certain schema—**’schema-on-read’** system.
- Every document must have a unique key, which can be auto-generated, but **it cannot be a number.**
- Collections cannot be nested. However, a document may hold a sub-collection attached to it in a coll-doc-coll-doc-coll-… etc. hierarchy with a maximal depth of 100.
- Special document data types: 1-D array, map (associative array), reference type. A reference is a path string leading to a document, e.g. a reference pointing to a document in a hierarchy. It may take the form coll_id/doc_id/coll_id/… etc. A possible problem with sub-collections is that they survive in an 'orphaned' state even if the parent collection is deleted.
- One-to-many relations can be implemented by embedding, **many-to-many** collections can be implemented by embedding combined with references.
- Firestore supports real time concurrent updates and atomic transactions. Queries are implemented via filters.
- A Realtime/snapshot listener generates an event when a document changes, and an application can subscribe to the event by a callback function[23] .

More on best practices: https://firebase.google.com/docs/firestore/best-practices

Designing, creating and loading our Firestore document store

Our planned queries:

[22] For a comparison see: https://firebase.google.com/docs/firestore/rtdb-vs-firestore#key_considerations

[23] https://firebase.google.com/docs/firestore/query-data/listen

- Listing of products and customers
- Submitting a new order with a single item
- Listing of the last 5 orders

Considering these queries, we'll design 3 collections: a flat Products, a flat Customer and a hierarchical Orders->Orderitems collection. Note that this would not suit a query about what was ordered and when, but such a query is not (yet) part of our application.

Alternatively, we could nest the Orders->Orderitems collection into the Customers collection. However, this would make it difficult to query e.g. a time-ordered list of the orders as the querying of sub-collections is normally **not allowed** without iterating the parent document collection.

Other considerations:

- We'll use automatic document ID's
- We'll order the orders with respect to date to support the listing of the last 5 items
- The CustomerID will be indexed automatically

We'll programmatically create and load the document store by reading from the Postgres database running on localhost and writing to the document store in the cloud.

We must create a GCP service account to access the Firebase store from our desktop machine[24].

1. Open your GCP project or start a new one. On the Databases -> Firestore page set: SELECT NATIVE MODE, then CREATE DATABASE (only one Firestore database is allowed in a GCP project)
2. Open the Firebase console: https://console.firebase.google.com/
    a. Create project -> Add Firebase to one of your existing projects. This is where we add a new Firebase project to the already existing GCP project. In fact these are two separate but connected projects. *Warning*: "Deleting a Firebase project deletes the Google Cloud project too, and all contained resources"
    b. Accept the Pay as you go option
    c. On the Firebase console -> Cloud Firestore. You can try to manually add/delete/edit collections and documents either here or at the GCP GUI-n.
    d. Create a collection called books with two book documents
    e. Firebase console Project overview -> cogwheel icon -> Project settings -> Service accounts tab -> Create service account -> Generate new private key. Here we can generate a json file that contains the token of the key of the automatically created service account. Save this file to a secure place because it allows access to the database.
    f. You can also find here a python code snippet for the connection.
3. At the conda prompt, switch to the virtual environment and type: `pip install firebase_admin`
   After this, you can connect to the database: `db = firestore.client()`

[24] We would need no service account to access the store from an app within our GCP project, but it is much easier and faster to develop and debug an app locally and then deploy it to GCP if needed. See also: https://cloud.google.com/compute/docs/access/create-enable-service-accounts-for-instances

a. Check that the Postgres service is up and the Northwind tables are available

b. Review and run the python main.py script in the flask_gcp_firestore folder that loads the 3 collections. This time you need no Flask web application, so you can start the script from the conda prompt by typing: `python main.py`[25]

c. Check the contents of the collections at the Firebase console.

PRACTICE: Review and run the new northwind.py web app that implements the order processing transaction using the Firestore database.

- Manually delete the demo order that was inserted by the main.py
- Implement the missing list_orders() function that returns a summary row for each order item of the last 5 orders. The result is rendered by the order_list.html template.

  The functionality that we must reproduce is as follows:

```
select o.orderdate::timestamp(0) without time zone as orderdate,
c.companyname, c.country, c.balance, p.productname, od.quantity, od.quantity
* od.unitprice as value, p.unitsinstock
from products p join orderdetails od on p.productid=od.productid
    join orders o on o.orderid=od.orderid
    join customers c on c.customerid=o.customerid
order by orderdate desc limit 5;
```

- You can use the same Html templates.
- Test the web app on http://localhost:5000/

Before you go…

- Shut down the project: IAM -> Manage resources

More info on developing for Firestore:

- Firestore programming reference: https://firebase.google.com/docs/firestore/manage-data/add-data
- https://medium.com/google-cloud/firebase-developing-an-app-engine-service-with-python-and-cloud-firestore-1640f92e14f4
- https://towardsdatascience.com/nosql-on-the-cloud-with-python-55a1383752fc

Creating a Firestore service:

- https://firebase.google.com/docs/firestore/quickstart#python_1

Comparing Firestore to Postgres

The table below shows a comparison of tools to enforce data integrity:

Postgres	Firestore
Schema is enforced in all DML operations	‘schema on read’

[25] If you receive an authentication error, this is probably due to the incorrect system time setting of the virtual machine. In the date and time setting system panel, change the system time to manual and then back to automatic to refresh the time.

Data types and user defined data types (like enumeration types)	Very limited set of data types e.g. no money or decimal type
Primary keys and foreign keys	The ID of a collection must be unique and can be automatically generated (like a serial type) but it cannot be a number. Foreign keys can be stored as attributes and enforced by security rules.
Unique constraints and check constraints	Security rules[26]
ACID transactional support	Transactions can be started from the client to prevent test-and-set type errors and atomicity is preserved by batched writes[27], but transactions cannot be combined with batched writes (see later).

DEMO: Logical errors in a concurrent environment

- Add a delay of 10 seconds after the checking of the stock and the balance (`time.sleep(10)`)
- Set the balance of all 3 customers to 10000 and the stock of the Chai product to 3 manually on the Cloud Firestore web console
- Start the app in 3 browser windows and order 2 units of Chai for each customer (value: 2*18=36). We expect that only *one order of the three can be successful due to the limited stock.*
- Verify that all 3 orders are processed with SUCCESS, all 3 customers are charged 36, and the new stock value is 1. We can see that without transactional control, *serious logical errors can occur*.

Using a Firestore transaction

A Firestore transaction may contain read operations followed by write operations. *If a transaction reads a document and then the document is modified by another client before commit, the transaction is rolled back and re-run automatically (retried 5 times).* However, the atomicity of the transaction is *not* ensured. Example:

```
# create a new customer called test manually and set its balance to 9
# then run this script in two parallel sessions

import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
import time

cred = credentials.Certificate("token.json")
firebase_admin.initialize_app(cred)
db = firestore.client()
transaction = db.transaction()
cust_ref = db.collection('customers').document('test')
```

[26] a client side app that uses the Firebase Admin SDKs runs with administrative privileges and bypasses the security rules defined for the Firestore db. We could only demonstrate security rules by a mobile client. Therefore, rules are not detailed in this text. For more information, see https://firebase.google.com/docs/firestore/security/get-started

[27] https://firebase.google.com/docs/firestore/manage-data/transactions#python_1

```
@firestore.transactional
def update_in_transaction(transaction, cust_ref):
    cust_ref_snapshot = cust_ref.get(transaction=transaction)
    new_balance = cust_ref_snapshot.get('balance') + 1
    print("New balance: ", new_balance)
    time.sleep(10)
    if (new_balance <= 10):
        transaction.update(cust_ref, {'balance': new_balance})
        print("Balance increased")
    else:
        print("Balance already maximal")

update_in_transaction(transaction, cust_ref)
```

Set the balance of the test customer to 9 manually and run the above script from two concurrent sessions. Though we receive the 'Balance increased' message from both sessions, the final balance will be 10 (and not 11) because one of the transactions (the one that also returned the 'Balance already maximal' message) was rolled back and re-run again. Therefore, *we were able to enforce the business rule that the balance cannot exceed 10*, not even in a concurrent scenario.

PRACTICE: Using the above code snippet as a template, add transactional control to the order_proc function in the Firestore-based python Flask application and test its correct operation in the scenario described in the Concurrency demo section. We expect that among several concurrent transaction*, only one* will run successfully, and the consistency of the database will be preserved. This means that after the end of all transactions, the stock will be correct, only one order will be accepted and only one customer's balance will be charged.

Batched writes

The atomicity of a batch containing multiple 'doc.set', 'update' or 'delete' operations can be enforced by the db.batch() object. Note that this solution does not provide a protection against external updates on documents already read by the client and thus *it cannot prevent 'test-and-set' type logical errors*[28].

It is not possible to nest batched writes within a transaction. Therefore, we must decide whether to ensure Atomicity or Isolation of the ACID requirements.

Transactions only work online i.e. they require a connection with the server, but batched writes can work offline as well.

DEMO: review and test the batched version (northwind_batch_write.py) of the demo app by injecting a runtime error, e.g. return 1/0, between two writes.

Some Firestore limits:

- A transaction or batch of writes can write to a maximum of 500 documents
- Read operations (get()) must precede write operations (set(), update())
- „In security rules for transactions or batched writes, there is a limit of 20 document access calls for the entire atomic operation in addition to the normal 10 call limit for each single document operation in the batch."

[28] Cf. "Firestore allows you to run sophisticated ACID transactions against your document data." https://cloud.google.com/firestore

- etc. …

As a general conclusion we can state that with NoSQL technologies the client (or the business logic layer) must support and implement all low level data processing operations, like joining collections etc. Transaction control and data integrity enforcement are also less robust compared to relational database technologies. Therefore, NoSQL technologies can be recommended for relatively ***simple business domains and processes****. An example is the blogger website.*

Other Firestore features of interest not detailed here

- real time updates
- offline data persistence

BigQuery overview and demo

“BigQuery is designed to query structured and semi-structured data using standard SQL”[29]

BigQuery was designed to run **read-only** OLAP-like queries on denormalized (flattened) data warehouse tables for decision support and business analysis. The native storage model is a compressed (and optionally partitioned) column store (called Dremel)[30]. All data is stored in encrypted form. BigQuery can also run queries directly on other data sources like Bigtable, Cloud storage or Google drive as well. For best performance, the data from such external sources should be first imported into the Dremel column store.

The logical data model is a Dataset that contains related tables with a fixed schema and typed fields, referenced like `project.dataset.table`

Mining the Northwind database

In this demo, we

- use the SQL Server version of Northwind
- create a denormalized view and export it to csv
- load the csv to a BigQuery table
- use the table to create a linear regression BigQueryML model
- assess the accuracy of the model

Steps:

1. Create a view that contains the value of the order items with the following fields: value_numeric, country, categoryid, p_unitprice, discount, pyear
2. Export the view to a csv file
3. Using a Chrome browser, create a new project: GCP main menu -> IAM&admin -> Manage resources
4. GCP main menu -> Analytics -> BigQuery, Enable API

[29] https://panoply.io/data-warehouse-guide/bigquery-architecture/

[30] https://medium.com/google-cloud/bigquery-explained-storage-overview-70cac32251fa

5. From the "…" menu of the project -> Create dataset, choosing the Multi-region called 'US (multiple regions in the US)'. You must specify an ID. Name the dataset 'northwind'. The dataset will be stored in encrypted form. Go to dataset.

6. Select the + create table, and specify Create table from upload -> schema auto-detect. Load the denormalized table and check the data types. Name the table 'nw'. Money type may be a problem. Go to table.

7. Review the schema and the preview data

8. It is a good idea to check the validity and contents of the source data before an analysis. Analytics -> Dataplex is a GCP service that supports this.

   a. From Dataplex, or straight from the BigQuery Studio (last tab next to the Schema tab), create and start an on-demand Profile task by clicking Quick data profile -> Details. This task will run in the background for ca. 5 minutes, then display the column statistics and histograms.

   b. Also, start a Data quality scan with a manually added quality rule (e.g. Freight >=0). You can also review and accept the 'de facto' rules based on the outcome of the profile task, e.g. the rule inferred for the Freight filed is "min: 0.02, max:1007.64". The administrative info related to the scan can be stored in the Northwind dataset. The results are accessible by clicking on the scan name on the Dataplex -> Data quality page, and selecting the Jobs history tab.

9. In BigQuery, SQL queries can be run in the console. Compose new query -> SQL console window

10. Create the model that predicts the value_numeric field from the country, categoryid, p_unitprice, discount, pyear fields using the following SQL statement (if you named the dataset 'Northwind' and the table 'nw'):

```
CREATE MODEL `northwind.nw_model` OPTIONS (model_type='linear_reg', input_label_cols=['value_n
umeric']) AS
SELECT value_numeric, country, categoryid, p_unitprice, discount, pyear --if  the  select  list
does not contain the class label, you get the error message ’Unable to identify the label column’
FROM `northwind.nw`
WHERE value_numeric is not null
```

11. Build the northwind.nw_model (takes ca. 30 s)[31], then Go to model. Check the Details and the Evaluation. You can see that the data set was split automatically for a training table (1702 records) and an evaluation table (453 records). The mean absolute error is 301.

12. We can evaluate the model using the full data set (though is not a recommended best practice in data science and we do it only for demonstrating the EVALUATE method). You can use the following command.

```
SELECT   * FROM ML.EVALUATE(MODEL `northwind.nw_model`, (
   SELECT value_numeric, country, categoryid, p_unitprice, discount, pyear
   FROM `northwind.nw`
))
```

[31] https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create

13. The results show that the average error of the prediction is 342, the explained variance is 53%, so the performance of the model is poor: the connection between the dependent variable and the independent variables is weak. If we re-create the model with the CALCULATE_P_VALUES = true, CATEGORY_ENCODING_METHOD='DUMMY_ENCODING' options, we should also see whether the contribution of each variable to the model is significant[32]. We can also query those records where the error was less than 40% using the following command. Similarly, we'll find that the worst prediction was 517.9 instead of 17.0.

```
SELECT * FROM ML.PREDICT(MODEL `northwind.nw_model`, (
   SELECT value_numeric, country, categoryid, p_unitprice, discount, pyear
   FROM `northwind.nw`
)) where abs((predicted_value_numeric-value_numeric))/value_numeric<0.4
```

PRACTICE: Modify the view to include a value_nominal field that has three categories for the low-price, middle-price and the high-price items. Try to predict the value_nominal field using the 'BOOSTED_TREE_CLASSIFIER' or the 'LOGISTIC_REG' methods. In how many cases/which percentage can you predict the class label correctly?

Some help:

```
select od.unitprice*od.quantity*(1-od.discount) value,
case when (od.unitprice*od.quantity*(1-od.discount)) < 200 then 'L'
       when (od.unitprice*od.quantity*(1-od.discount)) < 1200 then 'M'
       when (od.unitprice*od.quantity*(1-od.discount)) >= 1200 then 'H'
else 'N/A' end value_nominal,
c.CustomerID, c.CompanyName, c.Country, c.Balance, o.OrderID,  o.OrderDate, o.RequiredDate,
year(o.orderdate) pyear,
o.ShippedDate, o.ShipVia, o.Freight, o.ShipName, o.ShipAddress, o.ShipCity, o.ShipRegion,
o.ShipPostalCode, o.ShipCountry, od.UnitPrice, od.Quantity, od.Discount,
p.ProductName, p.SupplierID, p.CategoryID,
p.QuantityPerUnit, p.UnitPrice p_unitprice, p.UnitsInStock, p.UnitsOnOrder, p.ReorderLevel,
p.Discontinued
from Customers AS c join Orders AS o ON c.CustomerID = o.CustomerID join
[Order Details] AS od ON o.OrderID = od.OrderID join
Products AS p ON od.ProductID = p.ProductID
```

PRACTICE: Use the public bigquery-public-data.samples.natality[33] test table to predict the weight of a baby using linear regression. This table contains many records, so you can sample the record set with a WHERE RAND() < 0.0001 clause.

Some help:

- If your dataset was not created in the US, create a new dataset in the Multi-region called 'US (multiple regions in the US)' with the name us_dataset and then run the query below. This way you'll not need to set the region of the query manually (in the MORE -> Query settings -> Advanced options)

```
CREATE MODEL `us_dataset.natality_model`
OPTIONS
  (model_type='linear_reg',
    input_label_cols=['weight_pounds']) AS
SELECT
  weight_pounds,
```

[32] However, this feature does not seem to work

[33] https://cloud.google.com/bigquery-ml/docs/bigqueryml-natality

```
  is_male,
  gestation_weeks,
  mother_age,
  CAST(mother_race AS string) AS mother_race
FROM
  `bigquery-public-data.samples.natality`
WHERE
  weight_pounds IS NOT NULL AND RAND() < 0.0001


SELECT   * FROM   ML.TRAINING_INFO(MODEL `dataset_nev.natality_model`)


SELECT * FROM ML.PREDICT(MODEL `dataset_nev.natality_model`,
(select true as is_male, 40 as gestation_weeks, 30 as mother_age,'38' as mother_race))
```

Further reading

- Faculty training https://edu.google.com/programs/faculty/training
- Learning paths https://cloud.google.com/training
- Quickstarts , Short tutorials to help you get started with Cloud Platform products, services, and APIs https://cloud.google.com/gcp/getting-started
- Sample Projects https://go.google-mkto.com/CPI00b01C3FT0A2j1tciC0V
- GCP docs https://cloud.google.com/docs

7. Special data types

Graph modeling

In traditional database modeling, a UML Class Model or an Entity Relationship Model can be transformed into relational model by using

- one-to-one relationships for one-to-one associations and inheritance (parent/child specialization)
- one-to-many relationships for one-to-many associations, aggregation and composition
- many-to-many relationships for many-to-many associations

Then, the one-to-one and one-to-many relationships can be implemented in the relational model by foreign keys, while the many-to-many relationships are implemented by linking tables.

There is, however, a more natural and less restricted way to model and implement *all* of the UML/ERM features: the **graph concept** that only uses node types and edge types with their attributes in the schema, and node and edge instances in the database. In fact, the graph technology is a revival of an old database structure called the network database model.

Graph tables on SQL Server

In MS SQL Server 2022, graph and edge type tables can express many-to-many relationships in a more natural way than the traditional linking tables in the relational model. Graph tables can be used to model complex networks with various types of nodes[34].

- A 'graph' is a collection of graph and edge tables. We can have only one graph per database.
- Node tables contain the entities that we want to connect. A hidden $node_id is automatically generated for each record.
- Edge tables contain the edges connecting the nodes. It is a good practice to separate edges connecting certain types of nodes from others connecting other types, by placing them in separate edge tables. Edges can also be constrained with respect to the allowed target node types, so we can have a certain 'graph schema' of the network[35]. An edge may have attributes. A hidden $edge_id is automatically generated for each record. An edge is always directed $from_id $to_id.
- SQL queries may specify connection patterns to retrieve nodes or sub-graphs with certain connection features using the MATCH keyword. A pattern returns a sub-graph. The intersection of two sub-graphs can be selected using AND in the pattern specification.
- MATCH patterns may contain a SHORTEST_PATH construct that returns the shortest path between nodes[36].

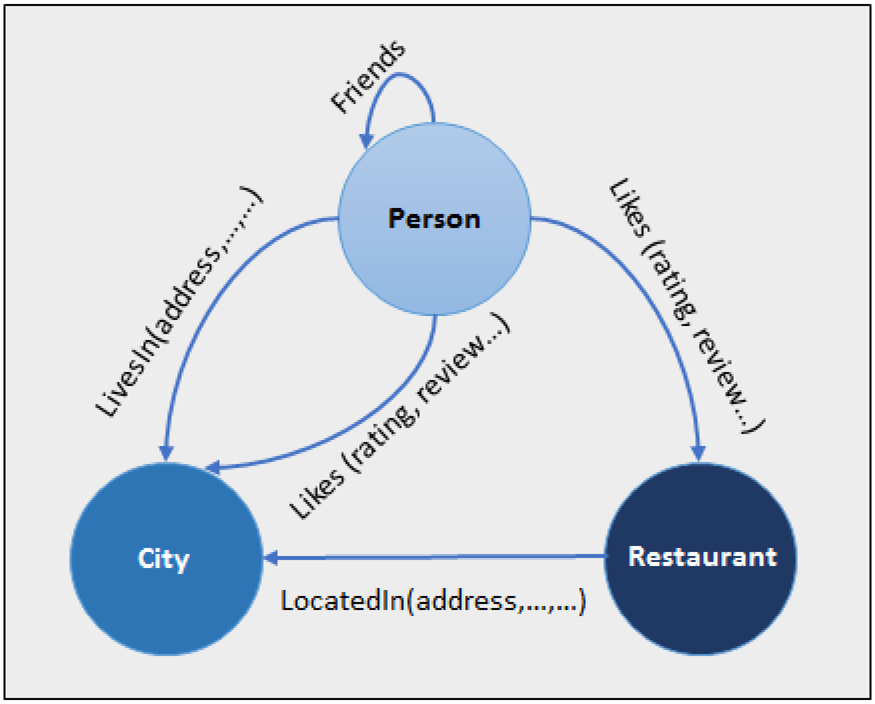
The example below uses a simple graph with three node types[37]:

[34] https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture?view=sql-server-ver16

[35] http://www.nikoport.com/2018/10/28/sql-graph-part-ii-the-edge-constraints/

[36] https://www.red-gate.com/simple-talk/sql/t-sql-programming/sql-graph-objects-sql-server-2017-good-bad/

[37] https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-sample?view=sql-server-ver15


Friends
Person
LivesIn(address,......)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)

```
--source: https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-
sample?view=sql-server-ver15
go
--use  graphdemo
go
       --drop table likes
       --drop table friendof
       --drop table livesin
       --drop table locatedin
       --drop table person
       --drop table restaurant
       --drop table city

-- create node tables
create table person (
  id integer primary key,
  name varchar(100)
) as node
create table restaurant (
  id integer not null,
  name varchar(100),
--  city varchar(100)
) as node
create table city (
  id integer primary key,
  name varchar(100),
  statename varchar(100)
) as node

-- create edge tables.
create table likes (rating integer) as edge
create table friendof as edge
```

```
--we constrain the friendof edge type to connect ONLY persons:
alter table friendof add constraint ec_1 connection (person to person)

create table livesin as edge
create table locatedin as edge

-- insert data into node tables. inserting into a node table is same as inserting into a
regular table
insert into person values (1,'john')
insert into person values (2,'mary')
insert into person values (3,'alice')
insert into person values (4,'jacob')
insert into person values (5,'julie')
insert into person values (6,'tom')

insert into restaurant values (1,'taco dell')
insert into restaurant values (2,'ginger and spice')
insert into restaurant values (3,'noodle land')

insert into city values (1,'bellevue','wa')
insert into city values (2,'seattle','wa')
insert into city values (3,'redmond','wa')
--select $node_id, * from city

-- insert into edge table. while inserting into an edge table,
-- you need to provide the $node_id from $from_id and $to_id columns.
insert into likes values ((select $node_id from person where id = 1),
       (select $node_id from restaurant where id = 1),9)
insert into likes values ((select $node_id from person where id = 2),
      (select $node_id from restaurant where id = 2),9)
insert into likes values ((select $node_id from person where id = 3),
      (select $node_id from restaurant where id = 3),9)
insert into likes values ((select $node_id from person where id = 4),
      (select $node_id from restaurant where id = 3),9)
insert into likes values ((select $node_id from person where id = 5),
      (select $node_id from restaurant where id = 3),9)
--select * from likes

insert into livesin values ((select $node_id from person where id = 1),
      (select $node_id from city where id = 1))
insert into livesin values ((select $node_id from person where id = 2),
      (select $node_id from city where id = 2))
insert into livesin values ((select $node_id from person where id = 3),
      (select $node_id from city where id = 3))
insert into livesin values ((select $node_id from person where id = 4),
      (select $node_id from city where id = 3))
insert into livesin values ((select $node_id from person where id = 5),
      (select $node_id from city where id = 1))

insert into locatedin values ((select $node_id from restaurant where id = 1),
      (select $node_id from city where id =1))
insert into locatedin values ((select $node_id from restaurant where id = 2),
      (select $node_id from city where id =2))
insert into locatedin values ((select $node_id from restaurant where id = 3),
      (select $node_id from city where id =3))

--delete friendof
-- insert data into the friendof edge.
```

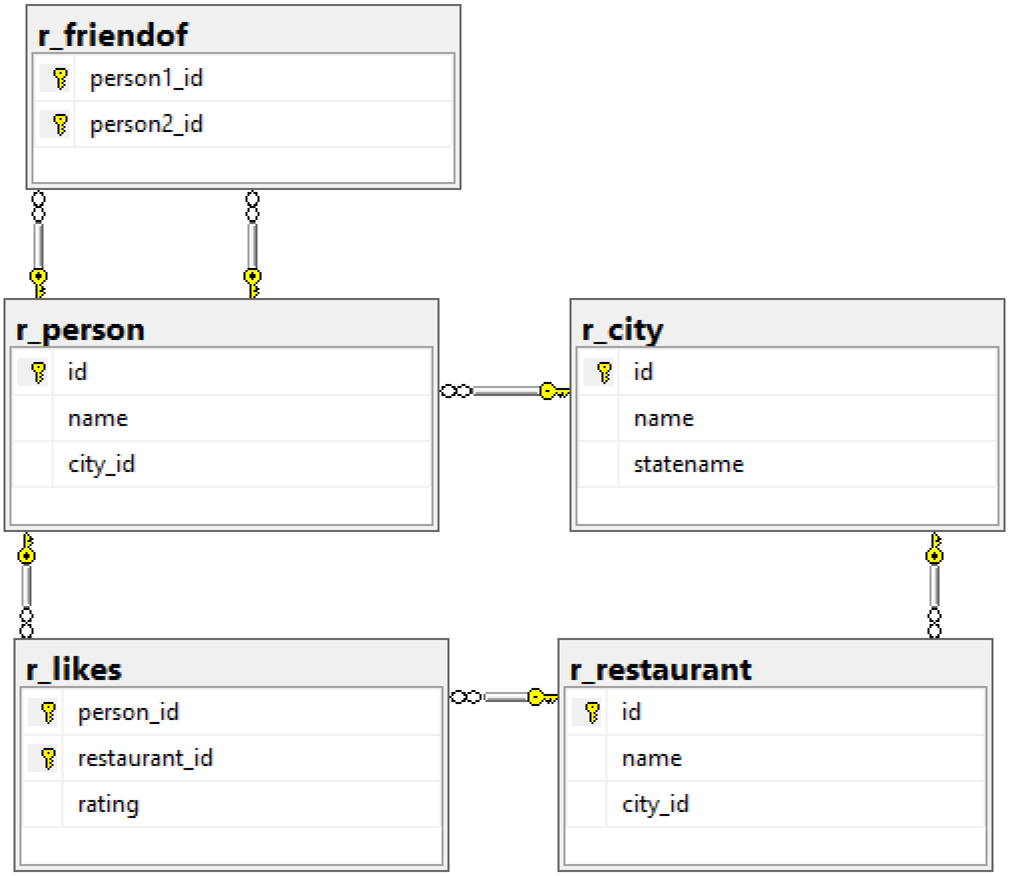
```
insert into friendof values ((select $node_id from person where id = 1), (select $node_id
from person where id = 2))
insert into friendof values ((select $node_id from person where id = 1), (select $node_id
from person where id = 5))
insert into friendof values ((select $node_id from person where id = 2), (select $node_id
from person where id = 3))
--friendof is a DIRECTED edge though friendship is undirected (mutual) --> no way to
express undirected relationships except using two edges
insert into friendof values ((select $node_id from person where id = 3), (select $node_id
from person where id = 2))
insert into friendof values ((select $node_id from person where id = 3), (select $node_id
from person where id = 5))
--repeated edge:
insert into friendof values ((select $node_id from person where id = 3), (select $node_id
from person where id = 6))
insert into friendof values ((select $node_id from person where id = 3), (select $node_id
from person where id = 6))

insert into friendof values ((select $node_id from person where id = 4), (select $node_id
from person where id = 2))
insert into friendof values ((select $node_id from person where id = 5), (select $node_id
from person where id = 4))

--repeated edges:
select * from friendof where json_value($from_id, '$.id') = 2 and json_value($to_id,
'$.id') = 5
--the internal node id indexing starts from 0, so instead of the edge 3->6, we specify 2-
>5
--we delete the repeated edge:
delete friendof where json_value($edge_id, '$.id') = 8
```

We can visualize friendships with Gephi, a free tool.

```
--------------------------------QUERIES--------------------------
-- friends of john
select p2.name
from person p1, person p2, friendof
where match(p1-(friendof)->p2)
and p1.name='john'

-- find restaurants that john likes
select restaurant.name
from person, likes, restaurant
where match (person-(likes)->restaurant)
and person.name = 'john'

-- find restaurants that john's friends like
select restaurant.name
from person person1, person person2, likes, friendof, restaurant
where match(person1-(friendof)->person2-(likes)->restaurant)
and person1.name='john'

--find people who like a restaurant in the same city they live in
select person.id, person.name
from person, likes, restaurant, livesin, city, locatedin
where match (person-(likes)->restaurant-(locatedin)->city and person-(livesin)->city)
```

```
--find people who have at least one friend who likes a restaurant in the same city (s)he
lives in
select distinct p1.id, p1.name
from person p1, person p2, friendof, likes, restaurant, livesin, city, locatedin
where match (p1-(friendof)->p2 and p2-(likes)->restaurant-(locatedin)->city and p2-
(livesin)->city)

-- find 2 people who are both friends with same person
--friendof is a DIRECTED edge though the relationship is undirected (mutual) --no way to
express undirected relationships except using two edges
select p0.name person, p1.name Friend1, p2.name Friend2
from person p1, friendof f1, person p2, friendof f2, person p0
--where match(p1-(f1)->p0<-(f2)-p2) and p1.id <> p2.id
where match(p1-(f1)->p0 and p2-(f2)->p0) and p1.id <> p2.id  --equivalent

--SHORTEST_PATH---------------------------------------------
--friends: all shortest paths from John
select p1.name,  p1.name+'->'+string_agg(p2.name, '->') within group (graph path) paths,
       last_value(p2.Name) within group (graph path) last_name
       ,count(p2.name) within group (graph path) depth
from person p1, person for path as p2,
    friendof for path as friend
--where match(shortest_path(p1(-(friend)->p2) {1,2})) and p1.name='john' --does not
contain the length 3 john->mary->alice->tom path
where match(shortest_path(p1(-(friend)->p2) +)) and p1.name='john' --contains all

--friends: the shortest path between John and Jacob
select name, paths, depth
from (
       select p1.name, p1.name+ '->'+string_agg(p2.name, '->') within group (graph path)
paths,
              last_value(p2.Name) within group (graph path) last_name
              , count(p2.name) within group (graph path) depth
       from person p1, person for path as p2,
              friendof for path as friend
       where match(shortest_path(p1(-(friend)->p2)+)) and p1.name='john' --and
p2.name='jacob' --ERROR: cannot select columns from FOR PATH tables
) a where last_name = 'jacob'
```

Note that the same domain could be modelled in the 'traditional relational' way with only 5 tables as follows, logically assuming that a person lives in a single city:



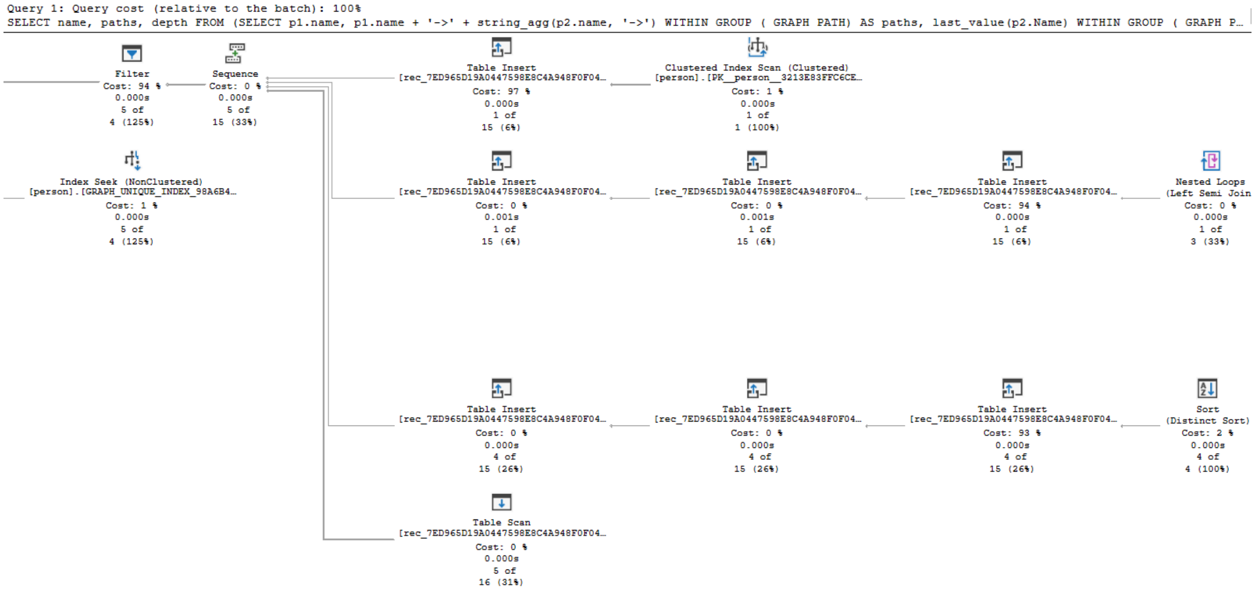
```
--drop table r_friendof
--drop table r_person
--drop table r_restaurant
--drop table r_city
go
create table r_city (id int primary key, name varchar(100), statename varchar(100))
create table r_person (id int primary key, name varchar(100), city_id int references
r_city)
create table r_restaurant (id int primary key, name varchar(100), city_id int references
r_city)
go
create table r_likes (
       person_id int not null references r_person,
       restaurant_id int not null references r_restaurant,
       rating int,
       constraint p_1 primary key (person_id, restaurant_id))
create table r_friendof (
       person1_id int not null references r_person,
       person2_id int not null references r_person,
       constraint p_21 primary key (person1_id, person2_id))
go
insert r_city values (1,'bellevue','wa'),(2,'seattle','wa'),(3,'redmond','wa')
insert r_person values (1,'john', 1), (5,'julie', 1), (4,'jacob', 3), (3,'alice', 3),
(2,'mary', 2)
```

```
insert r_restaurant values  (1,'taco dell', 1), (2,'ginger and spice', 2), (3,'noodle
land', 3)

insert r_likes values (1,1,9),(2,2,9),(3,3,9),(4,3,9),(5,3,9)
insert r_friendof values (1,2),(1,5),(2,3),(3,5),(4,2),(5,4)
go
use db_vi
-- friends of john
select p2.Name
from r_person p1 join r_friendof f on p1.id=f.person1_id join r_person p2 on
p2.id=f.person2_id
where p1.Name ='john'

-- find restaurants that john likes
select r.name
from r_restaurant r join r_likes l on r.id=l.restaurant_id join person p on
p.id=l.person_id
where p.Name ='john'

-- find restaurants that john's friends like
select r.name
from r_person p1 join r_friendof f on p1.id=f.person1_id join r_person p2 on
p2.id=f.person2_id
       join r_likes l on l.person_id=p2.id join r_restaurant r on r.id=l.restaurant_id
where p1.Name ='john'

--find people who like a restaurant in the same city they live in
select p.name
from r_restaurant r join r_likes l on r.id=l.restaurant_id join r_person p on
p.id=l.person_id
where r.id=p.city_id

--find people who have at least one friend who likes a restaurant in the same city (s)he
lives in
select p1.name
from r_person p1 join r_friendof f on p1.id=f.person1_id join r_person p2 on
p2.id=f.person2_id
       join r_likes l on l.person_id=p2.id join r_restaurant r on r.id=l.restaurant_id
where r.id=p2.city_id
```

PRACTICE: Implement the Orders, Order Details and Products tables of the Northwind database as a graph and write a T-SQL script that copies the contents into the new graph tables. Query the income by year and product name.

Performance

The actual execution plan of the shortest path from John to Jacob query shows an extensive use of temporary tables and nested loop operations, and also that the paths to all persons had to be computed before the one ending at Jacob could be selected—far from an ideal solution.



```
Query 1: Query cost (relative to the batch): 100%
SELECT name, paths, depth FROM (SELECT p1.name, p1.name + '->' + string_agg(p2.name, '->') WITHIN GROUP ( GRAPH PATH) AS paths, last_value(p2.Name) WITHIN GROUP ( GRAPH P…
```

PRACTICE: Write a short T-SQL script that generates a random graph database with a single node type (e.g. person) and a single edge type (e.g. friend_of). The number of nodes (N) and the average number of edges per node (E) should be parameterized. Measure the execution time of finding the *diameter* of the graph (i.e. the maximum of all shortest paths) in the function of the two parameters. Start with N=100, E=3 and increase N until the execution time exceeds 30 s. What do you expect and what do you measure?

Hints

- Use the WHILE <bool expression> BEGIN … END construct implement a cycle. You can find lecture notes about T-SQL on the Moodle pages.
- You can switch off row count messages and measure the actual execution time with the settings below. The CPU time can exceed Elapsed time on multi-processor systems. You should disable the statistics time/io while generating the database.

```
set nocount on
set statistics time on
set statistics io on
```

Further reading:

- https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/sql-server-2019-graph-database-and-shortest_path/
- https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/relational-databases/tables/graph-edge-constraints?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-shortest-path?source=recommendations&view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/queries/match-sql-graph?view=sql-server-ver16

GraphQL interfaces

The idea is to replace the joins in traditional SQL queries with the more intuitive graph based syntax. Functionality:

- Connecting to a database backend
- Parsing graphQL queries and checking schema conformance
- Executing the query on the backend and returning the results in graphQL format

More info:

- Specification of the graphQL server: https://spec.graphql.org/October2021/
- https://www.howtographql.com/graphql-python/0-introduction/
- https://blog.bitsrc.io/so-what-the-heck-is-graphql-49c27cb83754
- https://hasura.io/learn/
- https://cloud.hasura.io/projects

Driver={ODBC Driver for SQL Server};Server=adatb-mssql.mik.uni-pannon.hu,2019;Database=...;Uid=...

A native graph database: neo4j

Technical issues

Configuration, adding more memory:

- Safely shut down the virtual machine
- Increase memory to 8 GB
- Start the machine

Configuration, adding permissions:

- Step 1: Go to c:\Program Files\Neo4j Desktop.
- Step 2: Right click on Neo4j Desktop, select properties
- Step 3: Select 'Security' tab. On that select USERS
- Step 4: Click on Edit button and select USERS
- Step 5: In the Permission check the Allow Full Control box
- Step 6: Click on Apply button at the bottom.

Introduction

Neo4j is the most popular, open source, Java based graph database technology offering both desktop and cloud-based database servers[38]. The database files are stored in separate folders for each instance in the C:\Users\db\.Neo4jDesktop\relate-data\dbmss folder. Within the database folder, the persisted data files are located in the data\databases\neo4j folder. The data is stored in linked lists in the following data files[39]:

[38] https://neo4j.com/docs/getting-started/

[39] https://neo4j.com/developer/kb/understanding-data-on-disk/

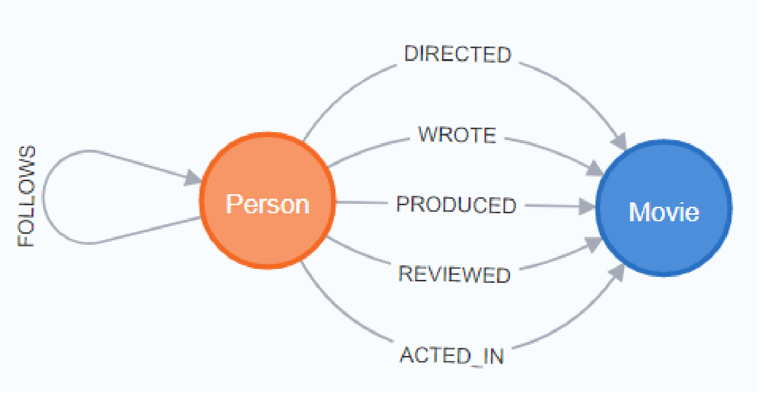
- Neostore.nodestore.* files: the nodes. In order to identify the nodes, there should be a unique property, e.g. ‘name’.
- Neostore.label.* files: Labels can be attached to nodes to group them together, e.g. all nodes representing an order could be labelled Orders. This resembles a node type, but a **node can have multiple labels**.
- Neostore. relationship.* files: the edges (a.k.a. relationships)
  - All relationships **must be directed** when we create them, but the direction of an edge **can be ignored** at query time.
  - Relationships can be indexed.
  - Relationships have a **single type** (=the single label of the relationship)
- Neostore. property.* files: the properties (attributes) attached to the nodes and edges. Properties are key-value pairs.

The type identifier of the attribute is stored for each attribute before the actual data in an attribute data block. The data block occupies 8 bytes, and the attribute record contains 4 blocks. The native data types are bool, float, integer, point, string, date/time and lists (arrays) of these. The PATH data type is an alternating sequence of nodes and relationships. Any type can take the NULL value.

The graph has an **‘optional’ schema** i.e. each node or edge may have different properties, though constraints can be added to force a domain model.

The Cypher query language

The database is managed in the query language **Cypher**[40]. The samples below use a demo database called Movies.



Cypher is built on patterns.

An example node pattern: `(:Movie {title: 'Forrest Gump', released: 1994})`

An example relationship pattern: `-[:ACTED_IN {roles: ['Forrest']}]->`

Some rules:

- An alias (reference variable) may be specified before the colon in either the node or relationship specifications, e.g. “p:Person”.

[40] https://neo4j.com/docs/cypher-manual/current/introduction/

- In a query using undirected relationships, the arrow has no head.
- Relationship types and property values are case-sensitive, though the Cypher language itself is not.
- The patterns can be combined in queries.

Some examples:

-
  ```
  MATCH (films:Movie) RETURN films.title    //all nodes having the Movie label
  ```

-
  ```
  MATCH (actor:Person)-[:ACTED_IN]-(film:Movie)
  WHERE actor.name='Tom Hanks'
  RETURN actor.name, film.title;         //list of movies in which T.H. acted in
  ```

-
  ```
  MATCH (actor:Person)-[:ACTED_IN]-(film:Movie),
  (director:Person)-[:DIRECTED]-(film:Movie)
  WHERE actor.name='Tom Hanks'
  RETURN actor.name, film.title, director.name; //as above, with the director
  ```

RETURN is a main statement. The typical main statements:

- RETURN: find and return the matching subgraph
- DELETE: delete nodes, relationships or paths
- REMOVE: remove properties from nodes and relationships, remove labels from nodes
- SET: update labels on nodes and properties on nodes and relationships
- CREATE: add nodes and relationships
- MERGE: try to match a pattern (that may include property values as well), if no exact match is found, create the pattern. Can be combined with RETURN

MERGE and CREATE examples:

-
  ```
  create (m:Movie {name: 'old movie'}) return *;      //returns the new node
  ```

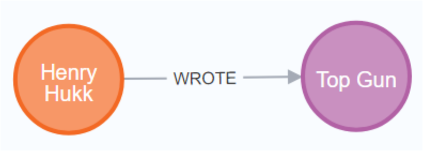
-
  ```
  merge (m:Movie {name: 'old movie'}) return *; //no new node created because an
  exact match was found
  ```

-
  ```
  create (m:Movie {name: 'old movie'}) return *;      //we have now 2 'old
  movie' nodes
  ```

-
  ```
  match (m:Movie {name: 'old movie'}) delete m;  //we deleted both of them
  ```

In the following example, we create a new Person and relate it to an existing movie via a new relationship, 'WROTE':

```
merge (p:Person {name: 'Henry Hukk'}) return p;
match (p:Person {name: 'Henry Hukk'}), (m:Movie {title: 'Top Gun'})
merge (p)-[:WROTE]->(m) return p,m;
```



A CREATE example:

```
CREATE
(:Person:Actor {name: 'Tom Hanks', born: 1956})     //this is a node with 2 labels
```

```
      -[:ACTED_IN {roles: ['Forrest']}]->     //roles are in an array
(:Movie {title: 'Forrest Gump', released: 1994})
      <-[:DIRECTED]-                          //note the direction of the arrow
(:Person {name: 'Robert Zemeckis', born: 1951})
```

This is how we can set or create a new property on a node:

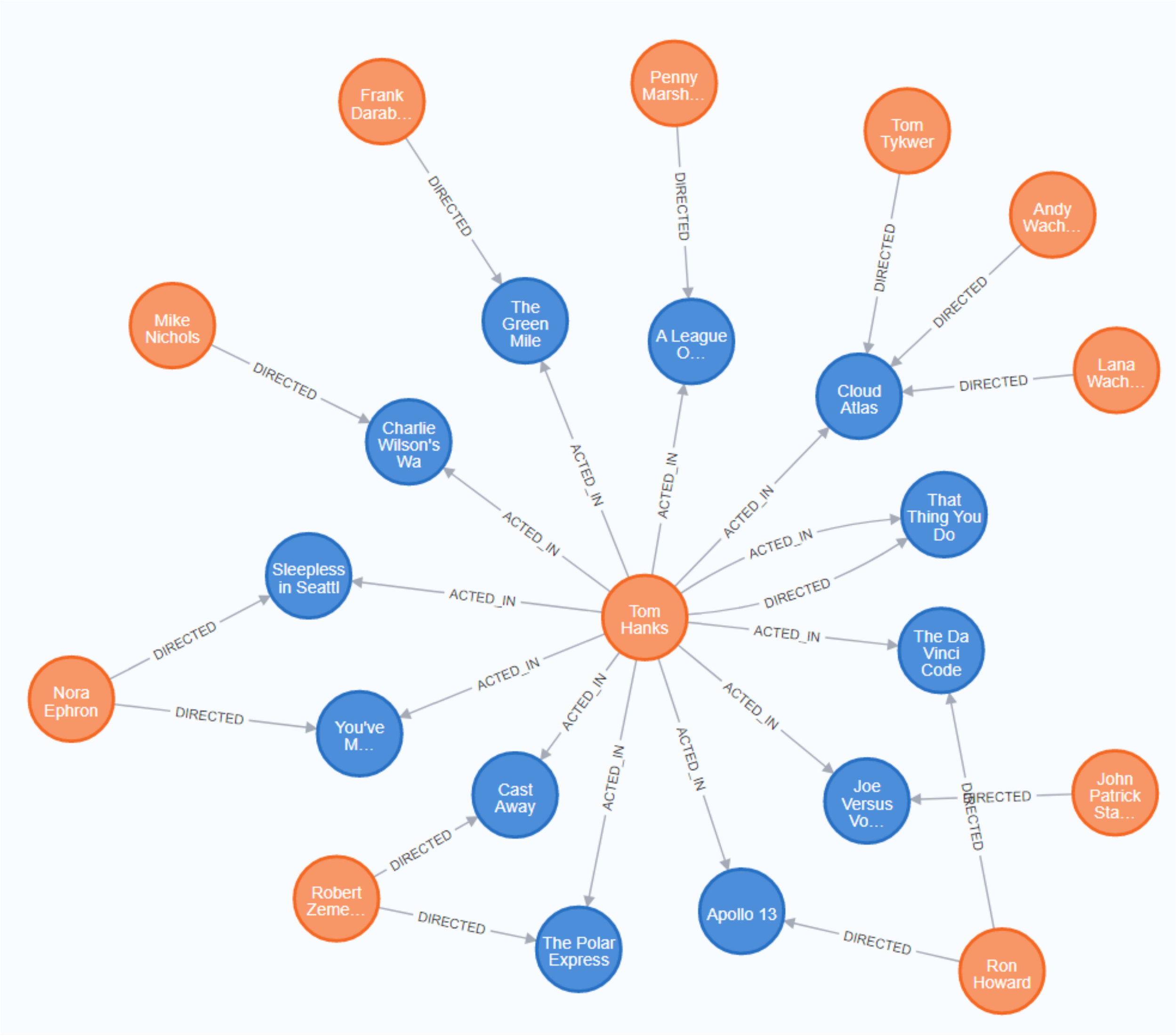
```
match (p:Person {name: 'Paul Blythe'})
set p.birthdate = date({year: 2018})  //will overwrite if existed
return p
```

An example for a property value based filter (find similar reviews):

```
match (p1:Person)-[r1:REVIEWED]->(m1:Movie), (p2:Person)-[r2:REVIEWED]->(m2:Movie)
where r1.rating=r2.rating
return p1.name, r1.rating, m1.title, p2.name, r2.rating, m2.title;
```

PRACTICE:

1. Open Neo4j Desktop and start a database instance connected to the Example project.
2. Import the sample Movies database from the database dump. Use the Open button associated with the load-movies.cypher file to open the dump script in the Neo4j browser.
3. At the neo4j$ prompt, type CALL db.schema.visualization() to review the node and relationship types
4. Try to run the above queries and check the results. If you RETURN the full nodes instead of the nodes' properties only, you can visualize the results also in the Graph view. For example using 'RETURN actor, film, director' in the third query results in this:



Grouping is performed implicitly by returning aggregated results. The group keys will be those properties on which we do not aggregate:

```
match (p1:Person)-[r1:REVIEWED]->(m1:Movie)
return m1.title, count(*), max(r1.rating), avg(r1.rating) as rat, min(r1.rating)
order by rat des
```

How many films do we have in the database:

```
match (:Movie) return count(*)
```

Finding a **shortest path**:

```
match p=shortestPath((p1:Person)-[:FOLLOWS*]-(p2:Person)) //p is a Path. Note the *
where p1.name='James Thompson' and p2.name='Paul Blythe'
return [n in nodes(p) | n.name] as stops
//returns: ["James Thompson", "Jessica Thompson", "Angela Scope", "Paul Blythe"]
```

The **diameter** of the graph along the undirected FOLLOWS dimension:

```
match (p1:Person), (p2:Person) where id(p1)>(id(p2)) //start and end cannot be the
same
match p=shortestPath((p1:Person)-[:FOLLOWS*]-(p2:Person))
return p, length(p) as length order by length desc limit 1
```

Finally, note that we have no schema. This is how to query the properties actually used in a graph:

```
match (p:Movie) return distinct  keys(p), size(keys(p))
match ()-[r:REVIEWED]->()  return distinct  keys(r), size(keys(r))
```

PRACTICE: Write and test these queries

- Query those persons who directed a movie in which they acted
- Find all the followers of the persons above
- Find the person who acted in the most movies which have a rating above 50
- Add a new hypothetical film to the graph that was directed by Ron Howard

Importing data to Neo4j

The easiest way to migrate a graph from SQL Server to Neo4j is via CSV files. This is an example for the graph version of the core Northwind tables.

1. Create suitable views that contain only the data we need in the graph

```
use northwind
create view vi_products as select productid, productname from products
go
create view vi_orders as select orderid, cast(orderdate as date) odate from orders
go
create view vi_orders_products as
select o.orderid, p.productid, od.Quantity quantity, od.Quantity*od.UnitPrice*(1-
od.Discount) price
from orders o join [Order Details] od on o.orderid=od.OrderID
             join products p on od.ProductID=p.productid
go
```

2. Export these views into CSV files using SSMS Export data tool
3. Copy the files into the import folder of the Neo4j database: C:\Users\db\.Neo4jDesktop\relate-data\dbmss\[DB id]\import
4. Load and create the orders, the products and finally the order items i.e. the edges among orders and products

```
load csv with headers from 'file:///products.csv' as line
create (p:product {id: line.productid, name: line.productname})
return p.id, p.name

load csv with headers from 'file:///orders.csv' as line
create (o:orders {id: line.orderid, odate: date(line.odate)})
return o.id, o.odate;

load csv with headers from 'file:///orders_products.csv' as line
match (p:product {id: line.productid}), (o:orders {id: line.orderid})
```

```
merge (o) -[c:CONTAINS
      {quantity: toInteger(line.quantity), price: toFloat(line.price)}]-> (p)
return c;
```

PRACTICE:

1. Stop the Neo4j database
2. Generate a random graph with 1000 nodes in SQL Server and measure the time needed to calculate the diameter
3. Export the graph to CSV files
4. Stop the SQL Server service and start the Neo4j database
5. Migrate the graph to Neo4j
6. Measure the time needed to calculate the diameter and compare it to the SQL Server performance

Storing images and BLOBs

- For single images of moderate size (such as images of merchandise in a web shop) the VARBINARY(MAX) data type is recommended:

```
insert my_table(image_column)
      select * from openrowset(bulk 'c:\my_path\my_photo.png', single_blob)
```

- For large video and other BLOB files, the FileTables technology is recommended on SQL Server. The file tables can be accessed both from the database and from other applications just like normal files stored in the file system[41].

PRACTICE: Implement a FileTable solution for storing the screen capture videos of the course.

[41] https://docs.microsoft.com/en-us/sql/relational-databases/blob/load-files-into-filetables?view=sql-server-2017

8. Acknowledgments

Some of the examples of this course notes are based on the book Dejan Sarka, Matija Lah, Grega Jerkič: Exam 70-463: Implementing a Data Warehouse with Microsoft SQL Server 2012, Microsoft Press, ISBN 978-0-7356-6609-2, 2015. Other examples are referenced in the footnotes.

9. APPENDIX A: SQL examples for self-learning

```
use NORTHWIND
select * from employees
select lastname, birthdate from employees

--the name of those customers who are located in London
select companyname, city
from customers
--where city LIKE 'L%' and (city LIKE '%b%' or city LIKE '%n%') --partial matching
where city IN ('London', 'Lander')
where city ='London' or city ='Lander'

where city IN ('London')
where city = 'London'

--who is the youngest employee? What is her name?
--1/2) the maximal birthdate
--aggregate functions: max, min, avg, std, sum, count
select max(birthdate) as max_year, min(birthdate) as min_year
--, lastname --would be an error
from employees

--2/2) embed this query
select lastname, birthdate from employees
where birthdate = ('1966-01-27 00:00:00.000')

select lastname, birthdate as "birth date" from employees
where birthdate = (
        select max(birthdate) as max_year
        from employees
)

--PROBLEM: find the ShipAddress of the first order
select orderdate, shipaddress from orders
where orderdate = (
        select min(orderdate) as min_date
        from orders
)

--ship addresses of the youngest employee
--joining tables
select distinct lastname, shipaddress
from orders o inner join employees e on o.employeeid=e.employeeid
where e.employeeid = (
        select employeeid from employees
        where birthdate = (
                select max(birthdate) as max_year
                from employees
        )
)
order by shipaddress --desc


--which products were ordered form the youngest employee
--note: always start with the FROM part of the query
select distinct p.productname, e.lastname
from orders o inner join employees e on o.employeeid=e.employeeid
        inner join [order details] od on od.orderid=o.orderid
        inner join products p on p.productid=od.productid
```

```
where e.employeeid=9  --she is the youngest
order by productname

--PROBLEM: which are the ship cities of products with CategoryID=1?
select distinct o.shipcity
from orders o inner join [order details] od on od.orderid=o.orderid
       inner join products p on p.productid=od.productid
where p.categoryid = 1  --our search conditon
order by shipcity


--No. of orders per employee?
--1/5) GROUPING
select employeeid, count(*)
from orders
group by employeeid

--a note aside: how to test for null?
select * from orders where employeeid is null
delete from orders where employeeid is null

--2/5) DO NOT DO THIS:
select e.lastname, count(*)
from orders o inner join employees e on o.employeeid=e.employeeid
group by e.lastname
--results in logical error if there are
--2 persons with the same lastname!!!

--3/5)
select e.lastname, e.firstname, count(*)
from orders o inner join employees e on o.employeeid=e.employeeid
group by e.employeeid, e.lastname, e.firstname
--this query misses the agent with no orders

--PROBLEM: list the number of products in each Category (we need the CategoryName also)
3. select c.categoryid, c.categoryname, count(*) as no_prod
1. from products p inner join categories c on p.categoryid=c.categoryid
2. group by c.categoryid, c.categoryname
4. order by no_prod desc

--4/5)
select e.employeeid, e.lastname, e.firstname, count(*)
from employees e left outer join orders o on o.employeeid=e.employeeid
group by e.employeeid, e.lastname, e.firstname
--problem: we have a fake count of 1 for the idle agent

--5/5)
select e.employeeid, e.lastname, e.firstname, count(o.orderid) as no_ord
from employees e left outer join orders o on o.employeeid=e.employeeid
group by e.employeeid, e.lastname, e.firstname
order by no_ord desc
--all problems solved

--who is whose boss
select e.lastname, boss.lastname as boss, bboss.lastname as boss_of_boss
from employees e left outer join employees boss on e.reportsto=boss.employeeid
       left outer join employees bboss on boss.reportsto=bboss.employeeid


--No. of orders per employee?
```

```
select e.lastname, count(orderid)
--count(*) would produce an order for Lamer who has no order at all
from employees e left outer join orders o on
--from employees e inner join orders o on
e.employeeid = o.employeeid
group by e.employeeid, e.lastname
order by count(*) desc

--who has no orders?
select e.*
from employees e left outer join orders o on
e.employeeid = o.employeeid
where o.orderid is null

--which is the biggest order?
--arithmetics

4. select o.orderid,
    cast(o.orderdate as varchar(50)) as order_date,
    str(sum((1-discount)*unitprice*quantity), 15, 2) as order_total,
    sum(quantity) as no_of_units,
    count(d.orderid) as no_of_items
1. from orders o inner join [order details] d on o.orderid=d.orderid
2. where...
3. group by o.orderid, o.orderdate
--order by o.orderdate
5. order by sum((1-discount)*unitprice*quantity) desc

--in order to order by date:  group by o.orderid, o.orderdate

--who's the most successful agent? with how many orders?
-- observe: having
-- count distinct
-- formatting numbers

select  u.titleofcourtesy+' '+u.lastname+' '+ u.firstname +' ('+u.title +')'  as name,
--select u.lastname as name,
str(sum((1-discount)*unitprice*quantity), 15, 2) as cash_income,
count(distinct o.orderid) as no_of_orders, count(productid) as no_of_items
from orders o inner join [order details] d on o.orderid=d.orderid
    inner join employees u on u.employeeid=o.employeeid
group by u.employeeid, u.titleofcourtesy, u.title, u.lastname, u.firstname

--having count(o.orderid)>200 -if we are only interested in agents with more than 200 orders
order by cash_income
--sum((1-discount)*unitprice*quantity) desc

--why do we have only 9?

select count(*) from employees

--it should be 10!
--we would also need those with 0 order
-- isnull function

select  isnull(u.titleofcourtesy, '')+' '+isnull(u.lastname, '')+' '+ isnull(u.firstname, '')
+' ('+isnull(u.title, '') +')'  as name,
isnull(str(sum((1-discount)*unitprice*quantity), 15, 2), 'N/A') as cash_income,
count(distinct o.orderid) as no_of_orders, COUNT(d.productid) as no_of_items
from employees u left outer join
    (orders o inner join [order details] d on o.orderid=d.orderid)
```

```
on u.employeeid=o.employeeid
--where u.titleofcourtesy='Mr.' –if we are only interested in men
group by u.employeeid, u.titleofcourtesy, u.title, u.lastname, u.firstname
order by sum((1-discount)*unitprice*quantity) desc

--which is the most popular product?
-- top 1

select top 1 p.productid, p.productname, count(*) as no_app,
    sum(quantity) as total_pieces
from products p left outer join [order details] d on p.productid=d.productid
group by p.productid, p.productname
order by no_app desc

--which agent sold the most of the most popular product?
--first version
 select top 1 u.titleofcourtesy+' '+u.lastname+' '+ u.firstname +' ('+u.title +')'  as name,
    sum(quantity) as no_pieces_sold
 from orders o inner join [order details] d on o.orderid=d.orderid
    inner join employees u on u.employeeid=o.employeeid
 where d.productid = 59 --we know this already
 group by u.employeeid, u.titleofcourtesy, u.title, u.lastname, u.firstname
-- having....
 order by sum(quantity) desc

/*************************************************************************
PROBLEM

--which agent sold the most of the most popular product, and what is the name of that product?
--in the pubs_access database: which is the most frequnted publisher of the author with the
most publications?

**************************************************************************/

--MULTI LEVEL GROUPING
--datetime fUNCTIONS
select 2
select getdate() --datetime data type
select DATEDIFF(s,'2013-10-10 12:13:50.370', '2013-10-10 14:16:50.370')
select DATEADD(s, 1000, '2013-10-10 14:16:50.370')
select YEAR(getdate()), MONTH(getdate())

--ORDERS BY MONTH AND AGENT
select e.employeeid, lastname, year(orderdate) as year, month(orderdate) as month,
count(orderid) as no_of_orders
from employees e left outer join orders o on e.employeeid=o.employeeid
group by e.employeeid, lastname, year(orderdate), month(orderdate)
order by lastname, year, month

--the same in another way:
select e.employeeid, lastname,
cast(year(orderdate) as varchar(4)) +'_'+  cast(month(orderdate) as char(2)) as month,
count(orderid) as no_of_orders
from employees e left outer join orders o on e.employeeid=o.employeeid
group by e.employeeid, lastname, cast(year(orderdate) as varchar(4)) +'_'+
cast(month(orderdate) as char(2))
order by lastname, month

--select case

select e.employeeid, lastname,
case
    when month(orderdate) < 10 then cast(year(orderdate) as varchar(4)) +'_0'+
cast(month(orderdate) as char(2))
    when month(orderdate) >= 10 then cast(year(orderdate) as varchar(4)) +'_'+
cast(month(orderdate) as char(2))
    else 'N.A'
end as month,
count(orderid) as no_of_orders
from employees e left outer join orders o on e.employeeid=o.employeeid
group by e.employeeid, lastname,
case
    when month(orderdate) < 10 then cast(year(orderdate) as varchar(4)) +'_0'+
cast(month(orderdate) as char(2))
    when month(orderdate) >= 10 then cast(year(orderdate) as varchar(4)) +'_'+
cast(month(orderdate) as char(2))
    else 'N.A'
end --a function serves better for this purpose
order by lastname, month

--using temp tables

select GETDATE() as ido into #uj_tabla
select * from #uj_tabla
drop table #uj_tabla

select * into #uj_tabla from employees


--drop table #tt

select e.employeeid, lastname, year(orderdate) as ev, month(orderdate) as month,
count(orderid) as no_of_orders
into #tt
from employees e left outer join orders o on e.employeeid=o.employeeid
group by e.employeeid, lastname, year(orderdate), month(orderdate)
order by lastname, month

select * from #tt

--Warning: Null value is eliminated by an aggregate or other SET operation.
--reason: an aggregate function(max,sum,avg..) exists on null values

select * from #tt

select lastname, str(avg(cast(rend_szam as float)), 15, 2) as avg_no_of_orders
--select lastname, avg(rend_szam) as avg_no_of_orders
from #tt group by employeeid, lastname
order by atlagos_rend_szam desc

--another solution for the same problem with an embedded query

select forras.lastname, str(avg(cast(forras.rend_szam as float)), 15, 2) as avg_no_of_orders
from (
        select e.employeeid, lastname, year(orderdate) as ev, month(orderdate) as month,
count(orderid) as no_of_orders
        from employees e left outer join orders o on e.employeeid=o.employeeid
        group by e.employeeid, lastname, year(orderdate), month(orderdate)
) as f -using an alias is compulsory
group by employeeid, lastname
order by avg_no_of_orders desc
```

```
--HOMEWORK PROBLEMS:

--AVG monthly number of orders for all products?

--Who had more than double order total compared to his boss?
```

10. APPENDIX B: Database administration and maintenance

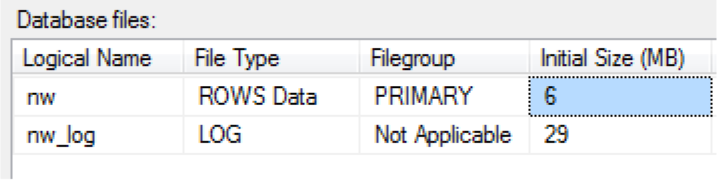
The most common tasks of relational database administration are as follows.

- Database file management
- Maintaining database performance—solving issues like file fragmentation and re-computing table statistics
- User and security management i.e. login roles, privileges, policies, and database encryption
- Implementing a backup strategy
- Configuring alerts for critical conditions

These elements may be implemented separately, or they may be combined in a single database maintenance plan.

Database files

Each SQL Server database must have at least two files associated with it, one containing the database objects (mdf file) and the other containing the transaction log (ldf file). It is the log file that is most critical with respect to failover recovery, so this file should be stored on redundant media, e.g. on a RAID array.



Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)
nw	ROWS Data	PRIMARY	6
nw_log	LOG	Not Applicable	29

In order to avoid fragmentation, the files are best placed on volumes that no other programs use. The initial size of the database can be estimated based on the planned application. Too small amounts of auto-grow may also lead to fragmentation.

In order to check database integrity it is a good practice to run DBCC CHECKDB ('DB_name') WITH NO_INFOMSGS, ALL_ERRORMSGS, as often as a full backup is created. Any output means a corruption of the database.

Database performance

Select the right recovery mode. Use the Full mode only if it is really needed by the application. Make sure that the following database options are set:

- Auto update statistics = TRUE
- Auto create statistics = TRUE
- Auto shrink = FALSE (if possible, do not use table or database shrinking at all)
- Page verify = CHECKSUM

Tuning a database for an application is beyond the scope of this course.

Backups

In a client-server database application, we use the **write-ahead transaction log** to tackle data loss situations. There are 3 types of errors for which we should be prepared:

1. The connection to the client is lost. -> We use the log to roll back all uncommitted transactions
2. The server process halts i.e. we lose the in-memory data and log buffers -> when the server restarts, it first rolls forward those transaction steps from the transaction log for which the relevant data pages were not flushed to the data files, then it rolls back all uncommitted transactions
3. The data file is lost -> backup the transaction log, use the previous full and differential backups and then the log backup to restore the database, then roll back all uncommitted transactions

Taking regular **backups** is a common way to prepare for disaster recovery. Backup files are to be stored on media different form that of the database data and log files. The three most common strategies:

#1. The minimum: use full backups in SIMPLE recovery mode[42], e.g. on a daily basis in off-load periods. The backups should be stored and re-written in a round-robin fashion e.g. over a weekly period. In this way the data loss can be limited to one day. It is a good practice to include in the backup also the master, model and msdb databases as well.

#2. Use full backups and differential backups in SIMPLE recovery mode, e.g. a daily full backup and differential backups every 2 hours. In this way the data loss can be limited to 2 hours.

#3. If the application requires that the possibility of data loss be minimized, the database must be set to FULL recovery mode and the second strategy must be combined with transaction log backups. Full recovery mode means that without backups, the log may grow substantially. An example strategy is a daily full backup, differential backups every 2 hours, and a transaction log backup every 20 minutes. If the data file is lost, all the committed transactions will be preserved.

Backups can be performed in the form of BACKUP DATABASE/RESTORE DATABASE SQL statements like this[43]:

```
USE [master]
RESTORE DATABASE [northwind] FROM  DISK = N'C:\Program Files\Microsoft SQL
Server\MSSQL14.PRIM\MSSQL\Backup\nw' WITH  FILE = 1,  NOUNLOAD,  REPLACE,  STATS = 5
```

PRACTICE: We simulate a disk error. We create a full backup of the Northwind database in a file device, stop the SQL Server PRIM service, delete the database file, start the service again and restore from backup, using the WITH REPLACE option. Check the contents.

For a regular backup, an option is to execute the BACKUP SQL statement in a scheduled job that we prepare manually. Alternatively, backup jobs can be integrated in a *database maintenance plan*.

Maintenance plans

Before the plan can be created, we must enable the use of extended stored procedures for SQL Server agent:

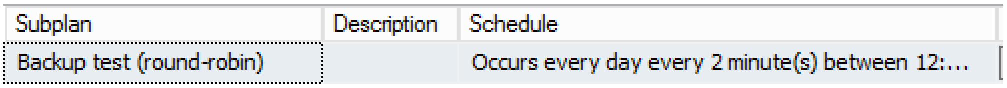
```
use master
sp_configure 'show advanced options', 1
go
reconfigure
go
sp_configure 'Agent XPs', 1
go
```

[42] The simple recovery mode means that he inactive parts of the log are regularly truncated.

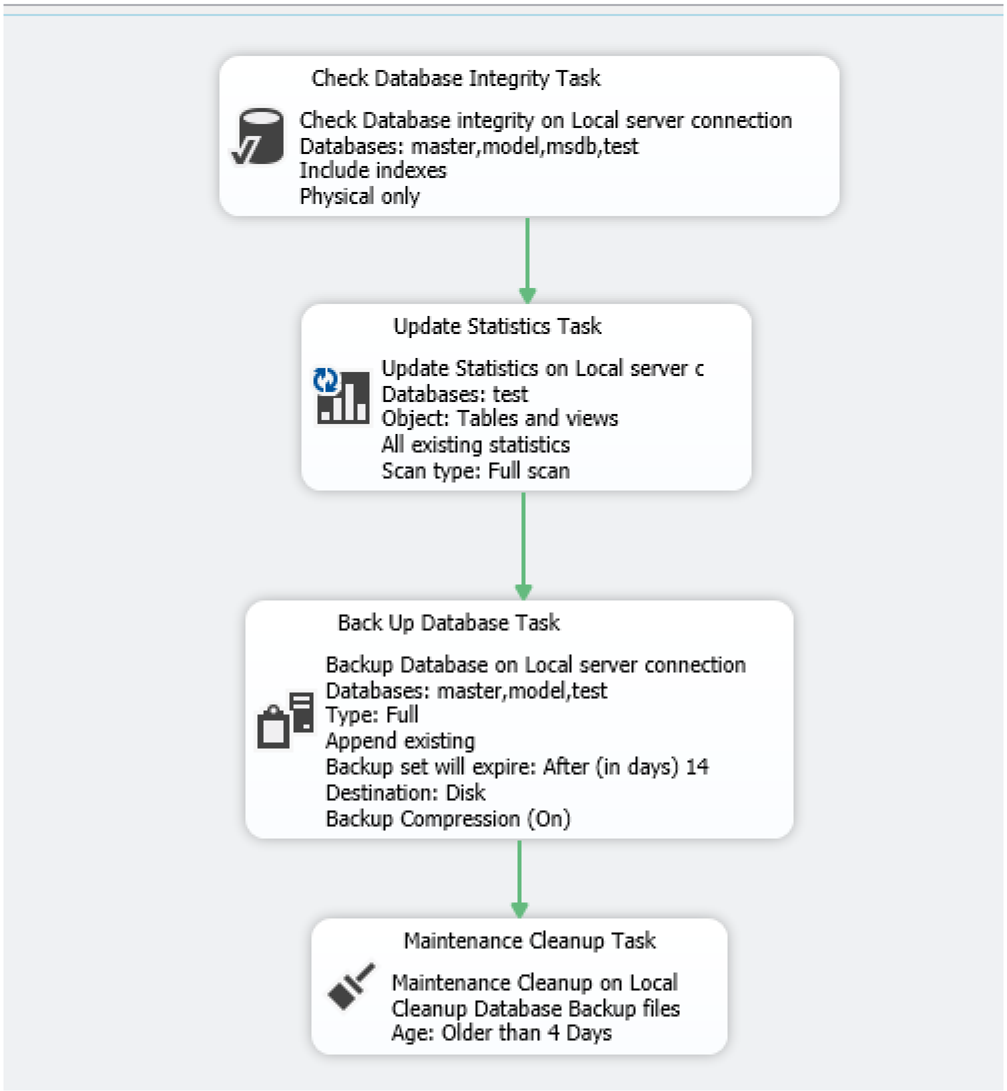
[43] SQL Scripts can be generated from each dialog panel of SSMS.

```
reconfigure
go
```

PRACTICE: Implement the backup strategy #1 in a new maintenance plan for a new test database by setting the schedule to every 2 minutes (for demo purposes). Also include index re-computing and integrity checking as tasks to be executed on a daily basis. Use Management -> Maintenance plans. Do not use the wizard. We set the 2-minute recurring interval only for demo purposes.



Subplan	Description	Schedule
Backup test (round-robin)		Occurs every day every 2 minute(s) between 12:...



Check Database Integrity Task

Check Database integrity on Local server connection
Databases: master,model,msdb,test
Include indexes
Physical only

Update Statistics Task

Update Statistics on Local server c
Databases: test
Object: Tables and views
All existing statistics
Scan type: Full scan

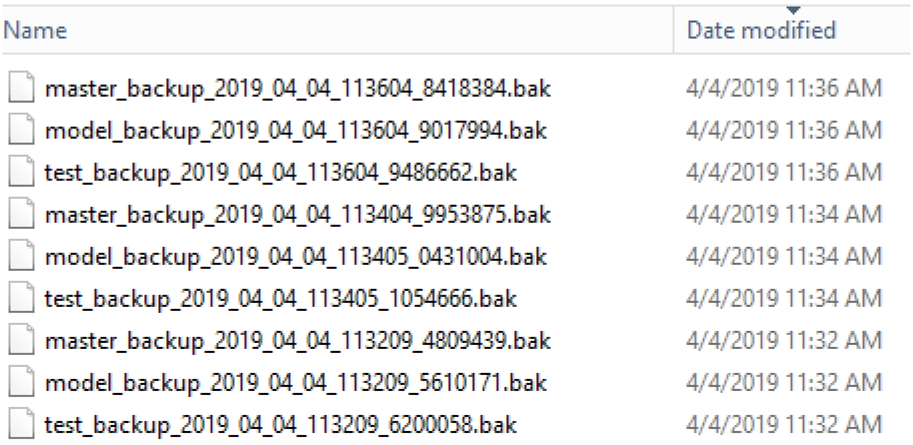
Back Up Database Task

Backup Database on Local server connection
Databases: master,model,test
Type: Full
Append existing
Backup set will expire: After (in days) 14
Destination: Disk
Backup Compression (On)

Maintenance Cleanup Task

Maintenance Cleanup on Local
Cleanup Database Backup files
Age: Older than 4 Days

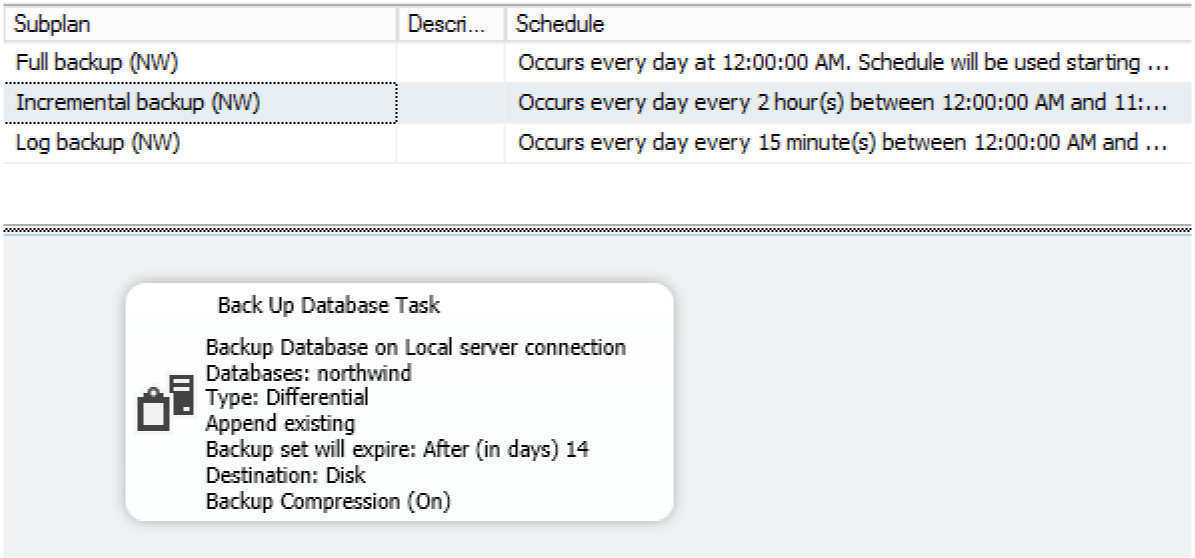
Check that the backups are generated every 2 minutes.



Name	Date modified
master_backup_2019_04_04_113604_8418384.bak	4/4/2019 11:36 AM
model_backup_2019_04_04_113604_9017994.bak	4/4/2019 11:36 AM
test_backup_2019_04_04_113604_9486662.bak	4/4/2019 11:36 AM
master_backup_2019_04_04_113404_9953875.bak	4/4/2019 11:34 AM
model_backup_2019_04_04_113405_0431004.bak	4/4/2019 11:34 AM
test_backup_2019_04_04_113405_1054666.bak	4/4/2019 11:34 AM
master_backup_2019_04_04_113209_4809439.bak	4/4/2019 11:32 AM
model_backup_2019_04_04_113209_5610171.bak	4/4/2019 11:32 AM
test_backup_2019_04_04_113209_6200058.bak	4/4/2019 11:32 AM

We cannot test the cleanup task because we cannot set a delete age less than a day.

PRACTICE: We implement the backup strategy #3 in a new maintenance plan for the Northwind database:



Subplan	Descri...	Schedule
Full backup (NW)		Occurs every day at 12:00:00 AM. Schedule will be used starting ...
Incremental backup (NW)		Occurs every day every 2 hour(s) between 12:00:00 AM and 11:...
Log backup (NW)		Occurs every day every 15 minute(s) between 12:00:00 AM and ...

Back Up Database Task

Backup Database on Local server connection
Databases: northwind
Type: Differential
Append existing
Backup set will expire: After (in days) 14
Destination: Disk
Backup Compression (On)

Alerts

It is a good practice to set up an alert for all severity 24 errors. Another classic critical condition is when the volume holding the database files is running low on free space.

PRACTICE: In this demo, we set up an **alert** that fires when the size of the Northwind database hits 150% of the current size. First we check the current size of the database in C:\Program Files\Microsoft SQL Server\MSSQL14.PRIM\MSSQL\DATA folder: 8.2 MB.

If the size is considerably larger, we first set the recovery mode to simple and then shrink the database i.e. free unused space, leaving 10%:

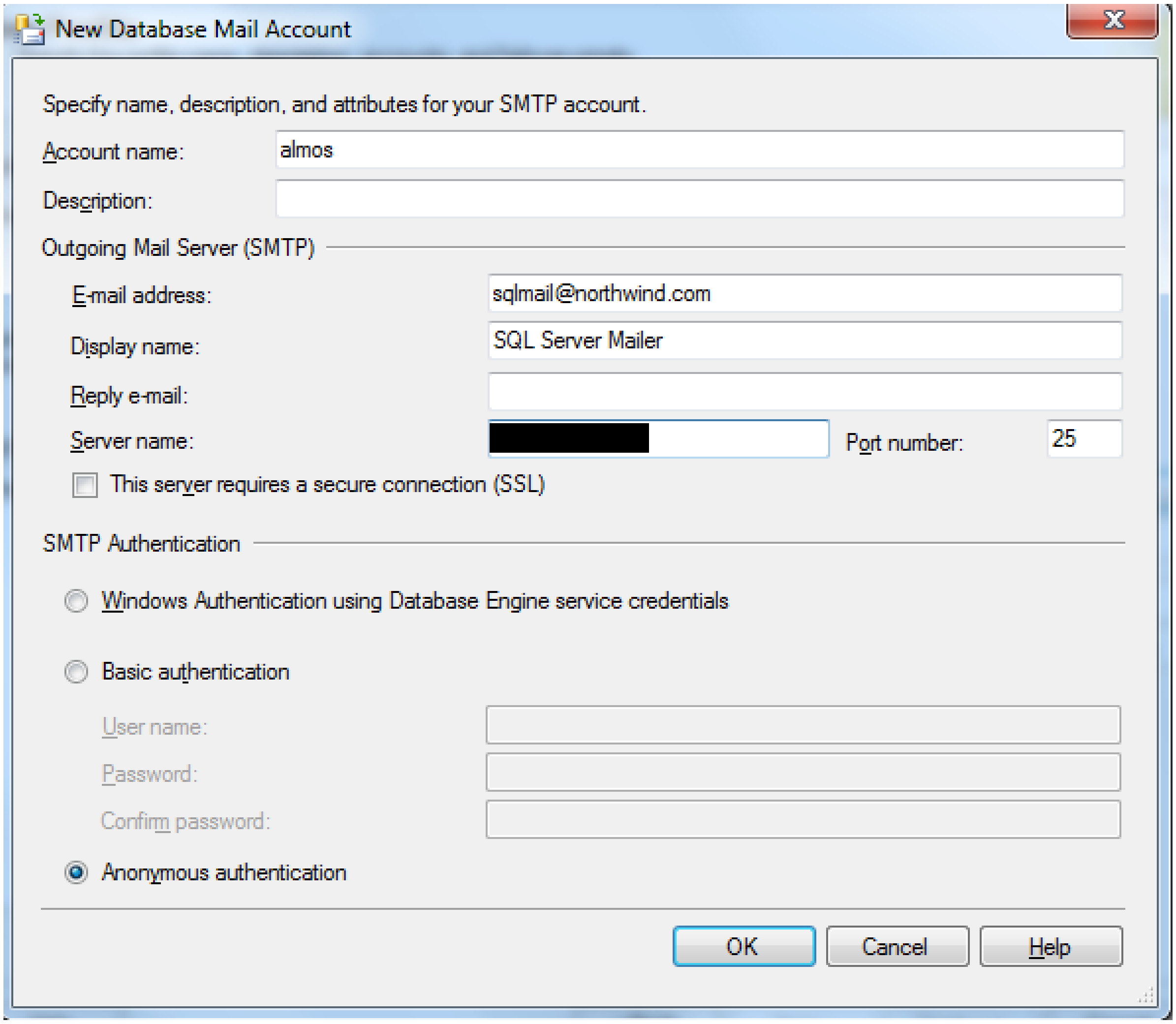
```
use master
dbcc shrinkdatabase(northwind, 10)
```

The process to set up the alert has 4 steps:

1. Set up a database mail profile
2. Enable the mail profile in SQL server agent that will run the alert
3. Create an operator (a person who will receive the alert and resolve the issue)
4. Create and test the alert

Setting up database mail

Select Server -> Management node -> Database mail ->Configure Database mail, then in the New profile panel, type prim_mail for profile name, then click Add SMTP account. Enter the name of our SMTP server.



New Database Mail Account

Specify name, description, and attributes for your SMTP account.

Account name: almos

Description:

Outgoing Mail Server (SMTP)

E-mail address: sqlmail@northwind.com

Display name: SQL Server Mailer

Reply e-mail:

Server name: [redacted] Port number: 25

□ This server requires a secure connection (SSL)

SMTP Authentication

○ Windows Authentication using Database Engine service credentials

○ Basic authentication

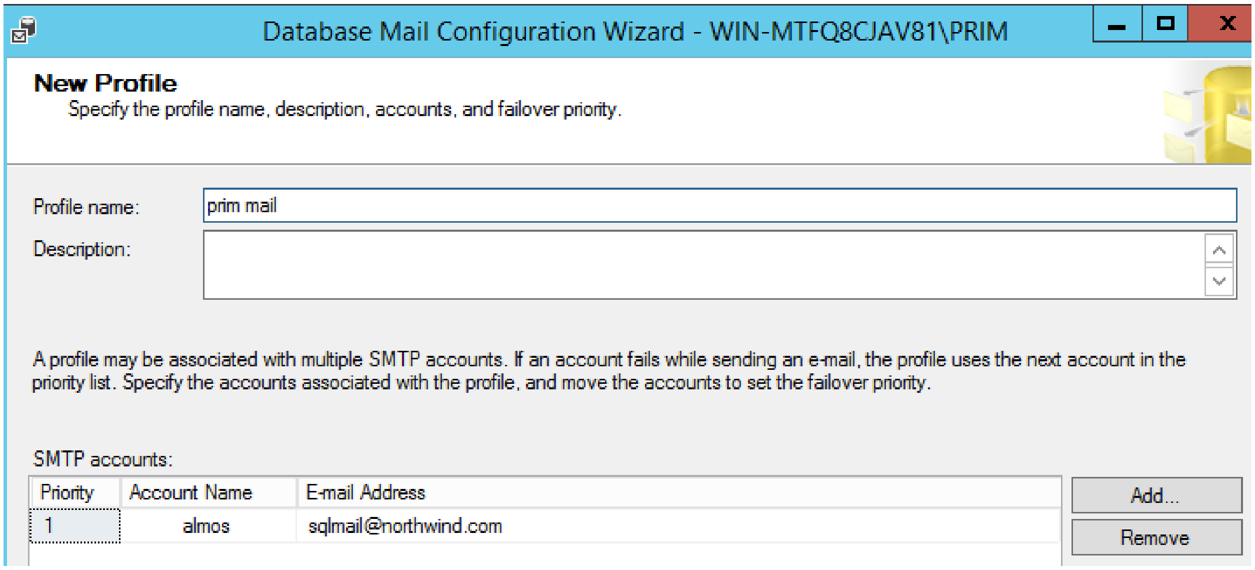
User name:

Password:

Confirm password:

● Anonymous authentication

Then you must specify the name of the new mail profile that will use the SMTP account:



Database Mail Configuration Wizard - WIN-MTFQ8CJAV81\PRIM

New Profile

Specify the profile name, description, accounts, and failover priority.

Profile name: prim mail

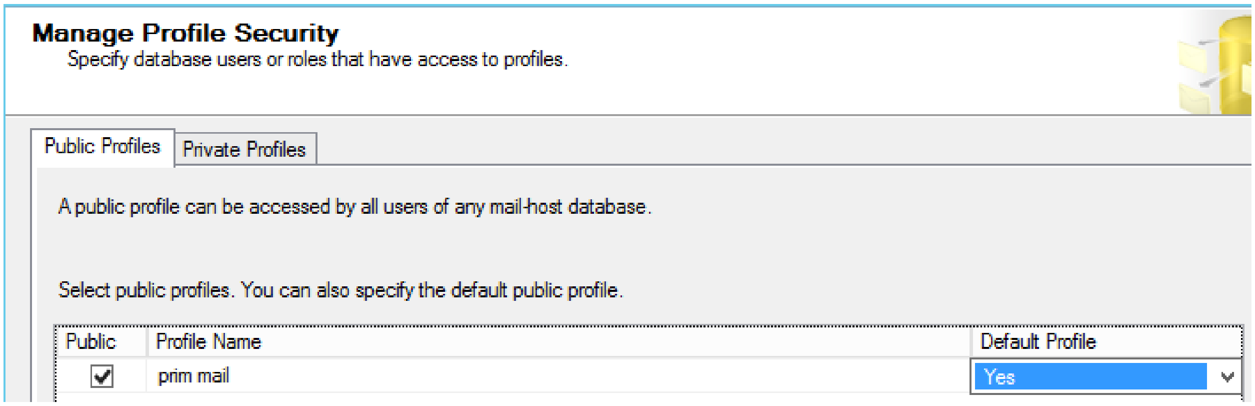
Description:

A profile may be associated with multiple SMTP accounts. If an account fails while sending an e-mail, the profile uses the next account in the priority list. Specify the accounts associated with the profile, and move the accounts to set the failover priority.

SMTP accounts:

Priority	Account Name	E-mail Address
1	almos	sqlmail@northwind.com

By making the profile public, every user can use it for sending emails:



Manage Profile Security

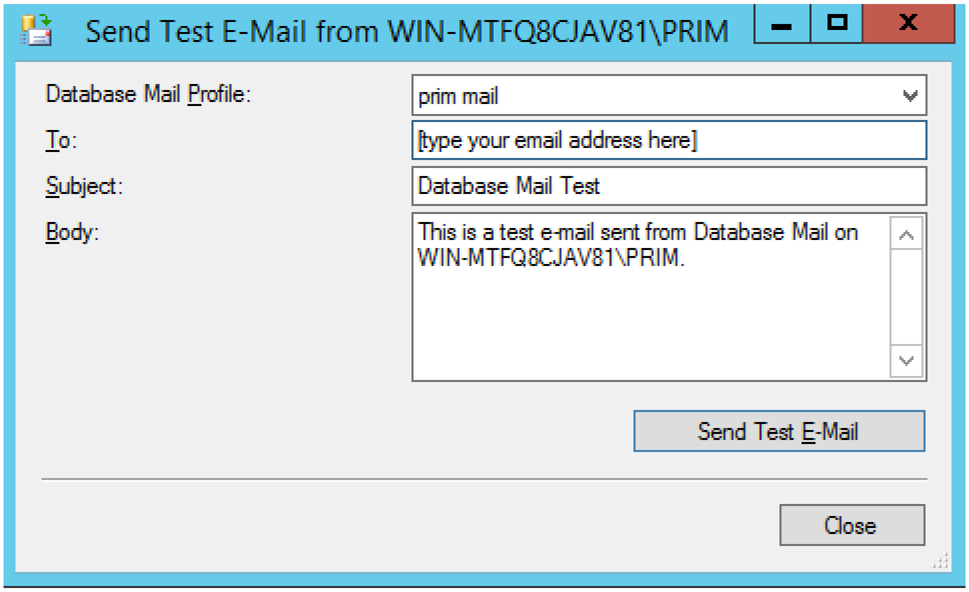
Specify database users or roles that have access to profiles.

A public profile can be accessed by all users of any mail-host database.

Select public profiles. You can also specify the default public profile.

Public	Profile Name	Default Profile
☑	prim mail	Yes

Verify that the profile works by selecting Database mail -> Send test E-mail. Check that the email is received as expected:



Database Mail Profile: prim mail

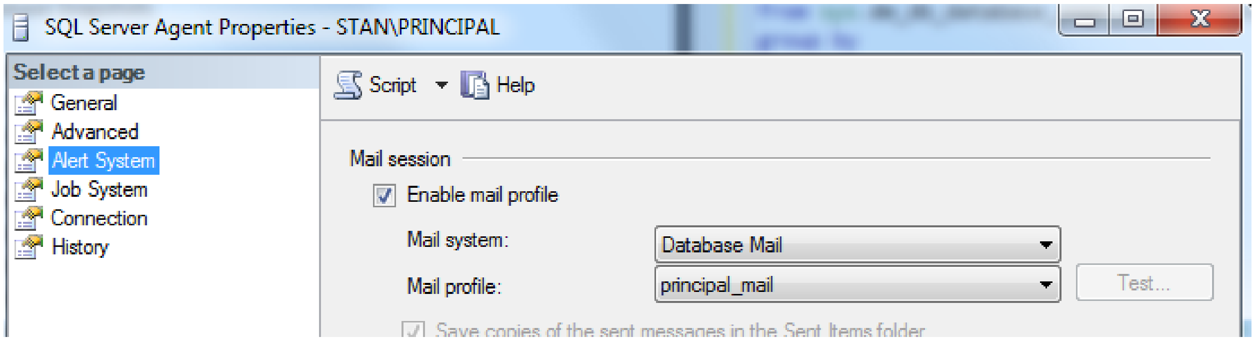
To: [type your email address here]

Subject: Database Mail Test

Body: This is a test e-mail sent from Database Mail on WIN-MTFQ8CJAV81\PRIM.

Enabling the mail profile in SQL server agent

Select Properties from the SQL server agent node popup menu:



☑ Enable mail profile

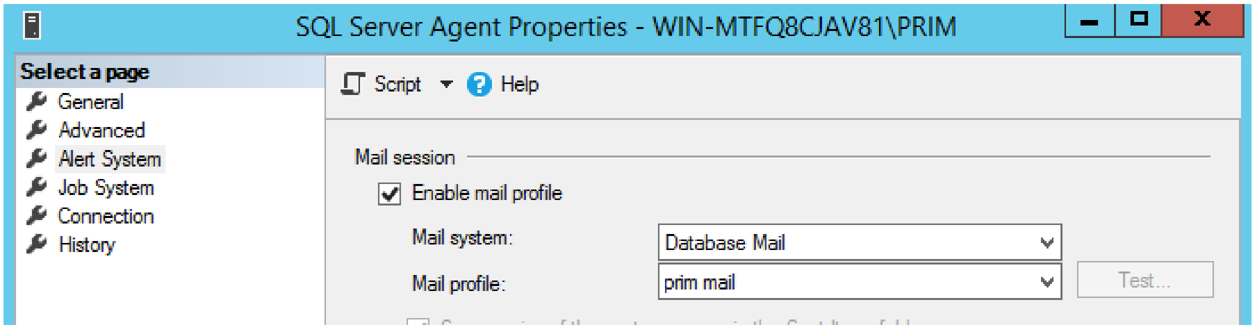
Mail system: Database Mail

Mail profile: principal_mail

☑ Save copies of the sent messages in the Sent Items folder

Creating an operator

Select Operators from the SQL server agent node and add a new operator. In the ‘E-mail name’ text box enter your own email address.



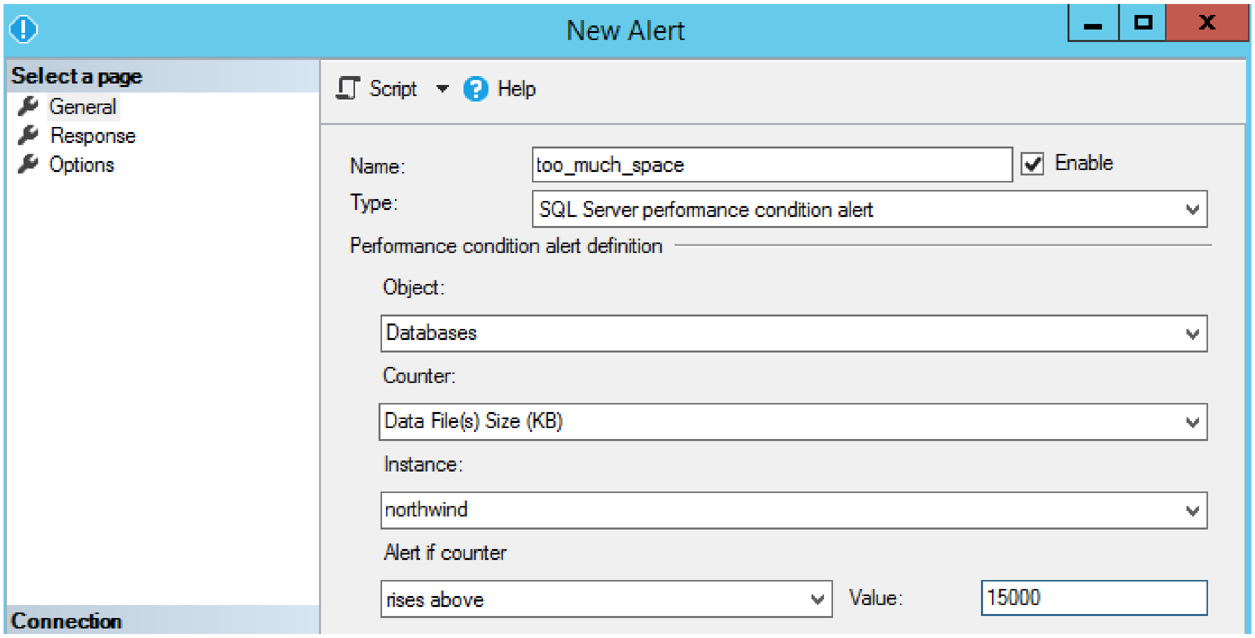
☑ Enable mail profile

Mail system: Database Mail

Mail profile: prim mail

Adding the alert

Select Alerts from the SQL server agent node and add a new alert:



Name: too_much_space ☑ Enable

Type: SQL Server performance condition alert

Performance condition alert definition

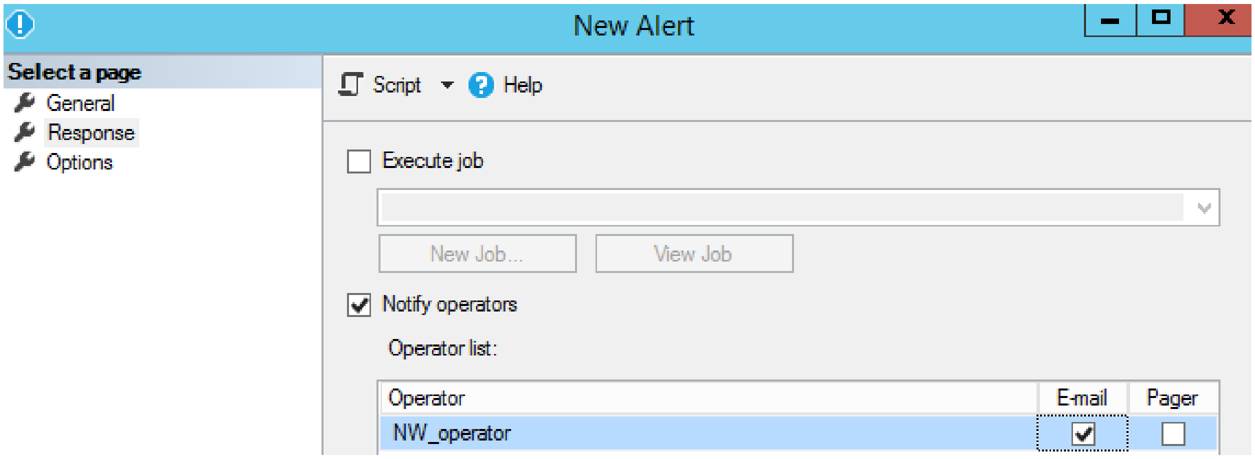
Object: Databases

Counter: Data File(s) Size (KB)

Instance: northwind

Alert if counter: rises above Value: 15000

Set the response to Notify operator (NW system administrator) by email:



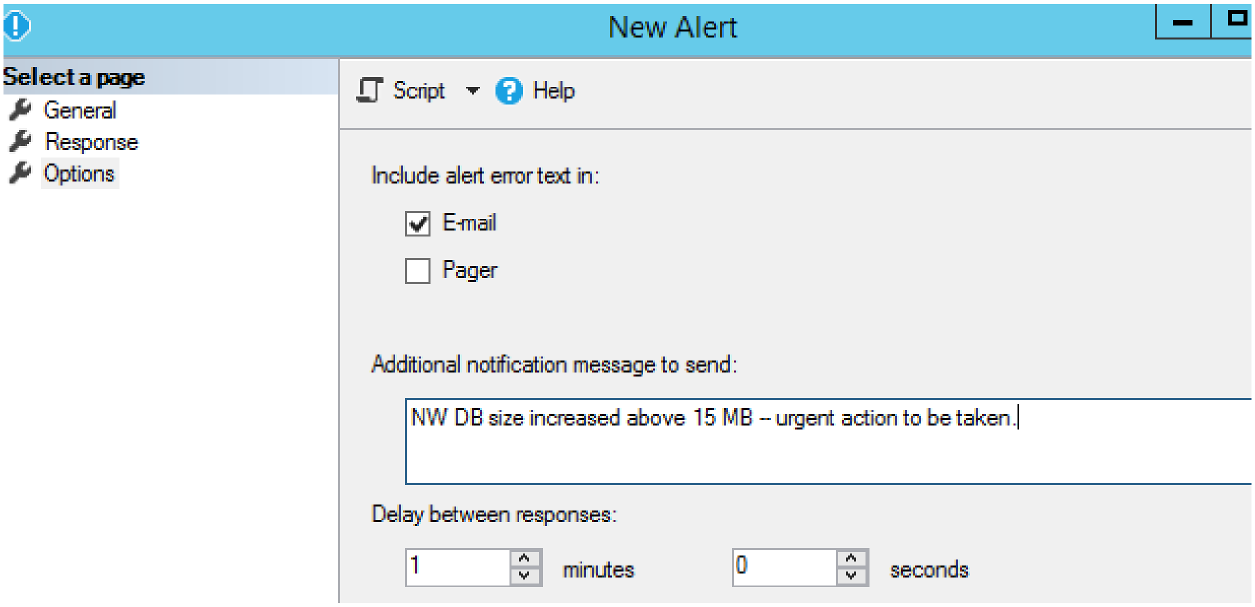
□ Execute job

☑ Notify operators

Operator list:

Operator	E-mail	Pager
NW_operator	☑	□

In the Options tab, add a custom message. Set the delay to 1 minute. *We use this setting only for demo purposes.* Note: If you set the Delay between responses to 0, the response will be repeated continuously as long as the alert condition is satisfied.



Include alert error text in:

☑ E-mail

☐ Pager

Additional notification message to send:

NW DB size increased above 15 MB -- urgent action to be taken.

Delay between responses:

1 minutes 0 seconds

Test the alert with the following script.

```
select  object_name(object_id) as 'tablename',
       count(*) as 'totalpages',
       sum(Case when is_allocated=0 then 1 else 0 end) as 'unusedPages',
       sum(Case when is_allocated=1 then 1 else 0 end) as 'usedPages'
from sys.dm_db_database_page_allocations(db_id(),null,null,null,'DETAILED')
group by
object_name(object_id)
--we will create a big table
go
create table big_table (a char(4000))
declare @i int=0
while @i<1000 begin
       insert big_table values ('a')
       set @i=@i+1
end
--big_table has 500 pages -> alert is fired
```

Check the alert history and your mailbox. The alert mails are coming repeatedly. *Do not forget to drop the big_table and disable or delete the alert*.

Overview of principals, privileges, schemas, roles

Overview of server **logins** vs. database **users** as principals. The login is linked to database users and it normally has a **default database**.

```
use master
create login nw_user with password='...', default_database=northwind
use northwind -context switch
create user nw_user for login nw_user
alter role db_datareader add member nw_user
```

PRACTICE: create a new database, a new login and a new database user

Overview of **privileges**, GRANT [which privilege] ON [which object] TO [whom], REVOKE, DENY.

```
use northwind
```

```
alter role db_datareader drop member nw_user
go
create procedure sp_list_employees
@city varchar(50) = 'London' –default value for the parameter
as
select * from Employees where City=@city
go
exec sp_list_employees
go
exec sp_list_employees 'Seattle'
go
grant execute on sp_list_employees to nw_user
go
```

Roles are groups of privileges, like read access to all tables.

Important **server level roles**: public, dbcreator, serveradmin, sysadmin

Important **database level roles**: db_owner, db_datareader, db_datawriter, negative roles: db_denydatawriter, db_denydatareader

```
use adworks
alter role exec_all_sp drop member nw_user
alter role db_owner add member nw_user
alter role db_denydatawriter add member nw_user

--check in a nw_user client connection:
use adworks
select * from Sales.Store
--(701 rows affected)
exec Person.sp_DeletePerson_Temporal 2
--(1 row affected)
--(1 row affected)
create table test (id int)
--Commands completed successfully.
insert test (id) values (25)
--Msg 229, Level 14, State 5, Line 10
--The INSERT permission was denied on the object 'test', database 'adworks', schema
'dbo'.
drop table test
--Commands completed successfully.
```

You can also create your own roles.

```
use northwind
revoke execute on sp_list_employees from nw_user
go
--this is how to grant execute for ALL objects of the database
grant execute to nw_user
--check in a client connection

--a new role
use adworks
create role exec_all_sp
grant execute on schema::Person to exec_all_sp--the role acts like a principal
--could also be: grant select, update on schema::Person to read_update_only_role etc.
create user nw_user for login nw_user
alter role exec_all_sp add member nw_user
```

```
--check in a client connection
use adworks
select * from Sales.Store
--Msg 229, Level 14, State 5, Line 8
--The SELECT permission was denied on the object 'Store', database 'adworks', schema
'Sales'.
exec Person.sp_DeletePerson_Temporal 2
--(1 row affected)
--(1 row affected)
```

PRACTICE: create a new role as a fusion of the db_datareader and db_datawriter roles, and assign the new user to this role. Test.

Database objects belong to a **schema** and the schema has a single **owner** (different from being a member of the db_owner database role). The owner can DROP the schema and create or drop objects in the schema.

Privileges are normally controlled at the schema/role level and not at the individual user/table level.

How to check which principals are in which roles?

```
select DP1.name as DatabaseRoleName, isnull (DP2.name, 'No members') as DatabaseUserName
from sys.database_role_members as DRM right outer join
sys.database_principals as DP1 on DRM.role_principal_id = DP1.principal_id left outer
join sys.database_principals as DP2 on DRM.member_principal_id = DP2.principal_id
where DP1.type = 'R'  --R: database role
order by DP1.name
```

Designing schemas and the associated roles and privileges for an application is an important part of relational modeling. Example: AdventureWorks database.

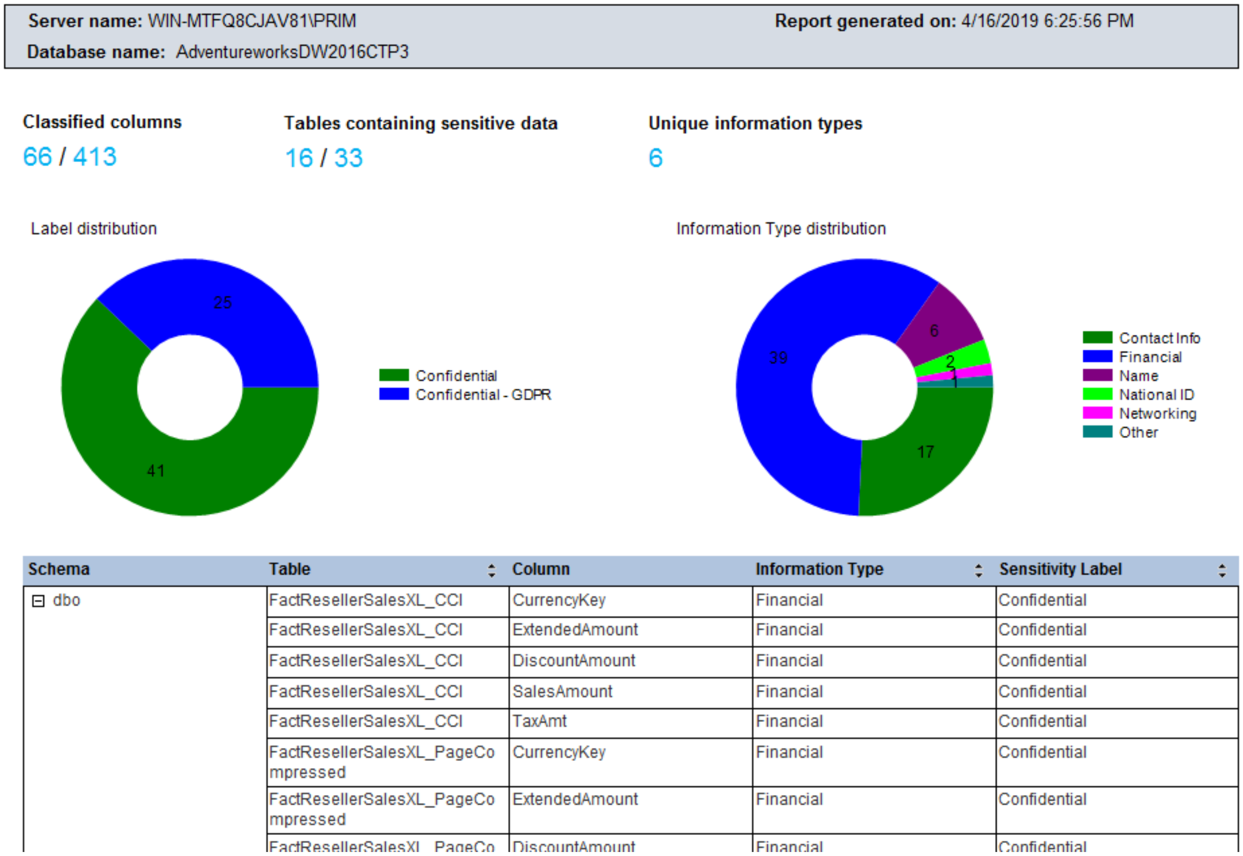
PRACTICE: design the schemas, roles and privileges for a big national library like OSzK

Data security in SQL Server

Sensitive data

In SQL Server 2017, fields can be classified automatically or manually w.r.t. info type (e.g. ‘Personal’) and sensitivity level (e.g. ‘Confidential-GDPR’).

Use the database Tasks -> Classify data tool. You can save the accepted recommendations. We can also use the Extended Properties catalog view called sys.extended_properties to query this information. This is what a sensitivity report looks like:



Server name: WIN-MTFQ8CJAV81\PRIM **Report generated on:** 4/16/2019 6:25:56 PM

Database name: AdventureworksDW2016CTP3

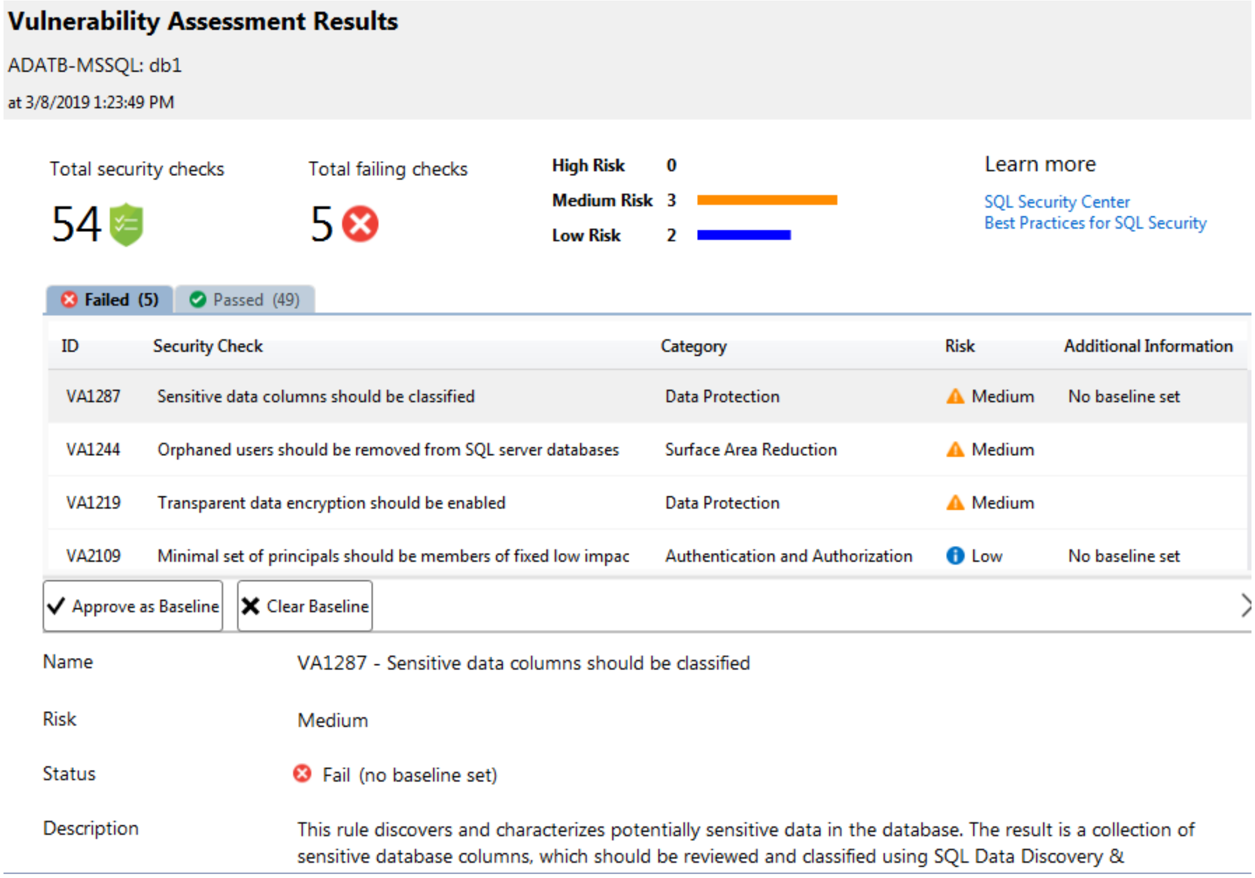
Classified columns 66 / 413

Tables containing sensitive data 16 / 33

Unique information types 6

Schema	Table	Column	Information Type	Sensitivity Label
⊟ dbo	FactResellerSalesXL_CCI	CurrencyKey	Financial	Confidential
	FactResellerSalesXL_CCI	ExtendedAmount	Financial	Confidential
	FactResellerSalesXL_CCI	DiscountAmount	Financial	Confidential
	FactResellerSalesXL_CCI	SalesAmount	Financial	Confidential
	FactResellerSalesXL_CCI	TaxAmt	Financial	Confidential
	FactResellerSalesXL_PageCompressed	CurrencyKey	Financial	Confidential
	FactResellerSalesXL_PageCompressed	ExtendedAmount	Financial	Confidential
	FactResellerSalesXL_PageCo	DiscountAmount	Financial	Confidential

You can also run an automatic vulnerability analysis:



Vulnerability Assessment Results

ADATB-MSSQL: db1

at 3/8/2019 1:23:49 PM

Total security checks

54

Total failing checks

5

High Risk	0
Medium Risk	3
Low Risk	2

Learn more

SQL Security Center
Best Practices for SQL Security

ID	Security Check	Category	Risk	Additional Information
VA1287	Sensitive data columns should be classified	Data Protection	Medium	No baseline set
VA1244	Orphaned users should be removed from SQL server databases	Surface Area Reduction	Medium	
VA1219	Transparent data encryption should be enabled	Data Protection	Medium	
VA2109	Minimal set of principals should be members of fixed low impac	Authentication and Authorization	Low	No baseline set

Name VA1287 - Sensitive data columns should be classified

Risk Medium

Status Fail (no baseline set)

Description This rule discovers and characterizes potentially sensitive data in the database. The result is a collection of sensitive database columns, which should be reviewed and classified using SQL Data Discovery &

Areas of data security

- Secure client connections -> **SSL**. A certificate to be used by the server must be installed.
- Database files and backup files (data at rest)
  - File level: **backup encryption** and **transparent data encryption** (TDE) that applies encryption to the data files as well
  - Field level: we can hide sensitive columns even from sysadmins ("**always encrypted**"), by using encryption and decryption in the clients
- Data being used by the server (data in motion)
  - **Dynamic data masking**: hide parts of sensitive columns via a mask defined in the schema e.g. ALTER COLUMN [Social Security Number] <data type> MASKED WITH (FUNCTION = 'partial(0,"XXX-XX-",4)')—show only the last 4 digits, and apply the XX pattern for the rest. Users with the UNMASK privilege can read the original value[44].

The recommended best practice is to apply field level protection to those fields classified as sensitive.

DEMO: masking

```
use northwind
--alter table employees drop column  SSN
--alter table employees drop column  email
alter table employees add ssn char(10) null, email varchar(200) --sensitive columns
```

[44] https://www.sqlshack.com/using-dynamic-data-masking-in-sql-server-2016-to-protect-sensitive-data/

```
alter table employees alter column ssn char(10) masked with (function =
'partial(1,".....",4)')
```

```
alter table employees alter column email char(200) masked with (function = 'email()')
go
update Employees set ssn ='1234567890'
update Employees set email ='sohase_mondd@citromail.hu'

select lastname, ssn, email from Employees --we can see all values

--however, if we create a read-write user
use master
create login test with password='h6twqPNO', default_database=northwind
go
use northwind --context switch
create user test for login test
alter role db_datareader add member test
alter role db_datawriter add member test
go
--switch to a connection of the test user
execute as user = 'test'
select lastname, ssn, email from Employees
revert
go
--this is what we see:
lastname      ssn           email
Davolio       1.....7890    sXXX@XXXX.com
Fuller        1.....7890    sXXX@XXXX.com

--we now give UNMASK privilege to test
grant UNMASK to test
go
execute as user = 'test'
select lastname, ssn, email from Employees
revert
go
--this is what we see:
lastname      ssn           email
Davolio       1234567890    sohase_mondd@citromail.hu
Fuller        1234567890    sohase_mondd@citromail.hu

--WARNING: the user can still modify the columns!!!
go
execute as user = 'test'
update Employees set email ='mondj_igent@citromail.hu'
select lastname, ssn, email from Employees
revert
go
revoke UNMASK from test
```

PRACTICE: try the default() and random() masking functions on the AdventureworksDW2016CTP3.dbo.DimCustomer table.

Configuring SSL

- create a new self-signed certificate with the certreq -new -f certparam.inf local.cer command, using the inf file to set the parameters

- use the mms console All tasks -> Manage private keys to add the MSSQL$PRIM user as a user with full control of the private key of the certificate
- in SS Config Manager use Protocols -> Properties to set the Enforce encryption option and restart the service
- in SSMS type select * from sys.dm_exec_connections and check the fields: encrypt_option=TRUE -> SSL, auth_scheme=SQL (the latter meaning SQL Server authentication)

Backup encryption

A Service Master Key (SMK) is created automatically when the instance is installed. In order to encrypt database backups, we need a Master Key that will be stored in the master database encrypted with the password specified and another copy encrypted with the SMK to enable automated use. This MK will be used to encrypt the private keys for the certificates created for database backups.[45]

DEMO: We create a master key:

```
use master
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'h6twqPNO'
go
--create a backup of the new MK
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'h6twqPNO'
go
--we use another password for the backup
BACKUP MASTER KEY TO FILE = 'C:\install\exportedmasterkey'
    ENCRYPTION BY PASSWORD = 'h6twqPNOh6twqPNO'
go

--we create a private-public key pair and a self-signed certificate
CREATE CERTIFICATE backup_cert_master   --this will be encrypted with the MK
   WITH SUBJECT = 'NW DB backup',
   EXPIRY_DATE = '20301031'

--it is wise to create a backup of it for disaster
BACKUP CERTIFICATE backup_cert_master TO FILE = 'C:\install\exportedcert'
--you can restore it by CREATE CERTIFICATE …

--we back up the database
BACKUP DATABASE northwind TO DISK = 'C:\install\nw_enc.bak'
WITH COMPRESSION, ENCRYPTION (
   ALGORITHM = AES_256,
   SERVER CERTIFICATE = backup_cert_master
), STATS = 10
GO
```

Try to restore the encrypted backup set to a new database on PRIM. The backup will be decrypted automatically using the `backup_cert_master`. Then try to do the same on SECOND. The contents of the backup will be unreadable.

PRACTICE: use the master key to back up the test database. Try to restore it on the THIRD server.

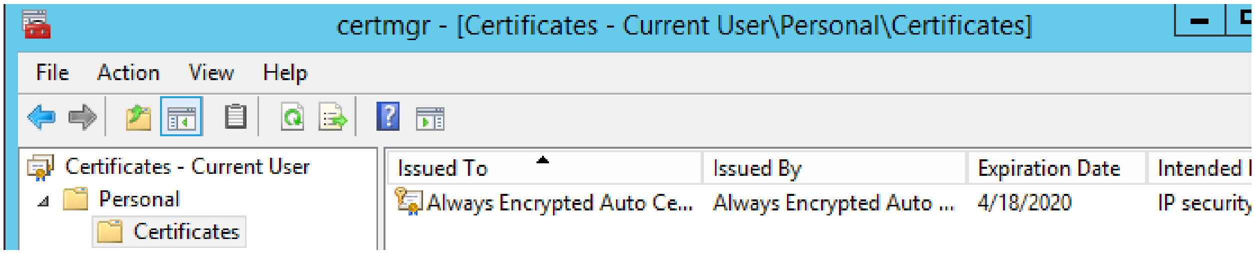
[45] https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/backup-encryption?view=sql-server-2017

Always encrypted

The encryption/decryption occurs in the client and the keys are stored outside the server instance (either locally in the Windows certificate store or in Azure Key Vault).

DEMO: we add Always Encrypted to the Address column of the Employees table.

1. Select Northwind -> Tables -> Employees -> Encrypt columns and select the Address field
2. Choose Randomized type. This means that grouping, filter by equality, and joining tables on encrypted columns will not be possible on the server, but an ECB[46] type attack against the column is prevented
3. Choose to generated a new column key
4. Accept that the collation will be changed to binary
5. On the next pane, select the Windows certificate store. This is outside the SQL server instance so the SQL server administrator in the sysadmin server role will have no access to it.
6. Check the new certificate in the certmgr.msc mmc application:


certmgr - [Certificates - Current User\Personal\Certificates]
File
Action
View
Help
Certificates - Current User
Personal
Certificates

Issued To	Issued By	Expiration Date	Intended I
Always Encrypted Auto Ce...	Always Encrypted Auto ...	4/18/2020	IP security

7. SELECT * from the table to verify the result—this is what the sysadmin will see
8. If you want to see the decrypted results, open a new connection and in the Options, select Additional Connection Parameters and type "Column Encryption Setting = Enabled".
9. When you run a query, you should enable parameterization in order to support inserts, updates or filtering of encrypted columns.

PRACTICE: select the Deterministic encryption type for the Title column and verify that the same ciphertext will be generated for the same plaintext, making an ECB attack possible.

[46] https://searchsecurity.techtarget.com/definition/Electronic-Code-Book