Міністерство освіти і науки України
Державний університет «Житомирська політехніка»

Затверджено науково-методичною
радою ЖДТУ
протокол від «30» травня 2018 р. №5

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

для виконання лабораторних робіт з навчальної дисципліни
«Адміністрування баз та сховищ даних
для студентів освітнього рівня «БАКАЛАВР»
денної форми навчання
спеціальності 123 «Комп'ютерна інженерія»
освітньо-професійна програма «Комп'ютерна інженерія»
факультет інформаційно-комп'ютерних технологій
кафедра комп'ютерної інженерії та кібербезпеки

Обговорено на засіданні кафедри
комп'ютерної інженерії та кібербезпеки

Протокол № 5
від „30" січня 2018 р.

Розробники:
к.т.н., зав. кафедри компютерної інженерії та кібербезпеки А.А. Єфіменко
к.пед.н., доцент кафедри інженерії програмного забезпечення
В.Н. Ковальчук
к.т.н., доцент, доцент кафедри інженерії програмного забезпечення
І.І. Сугоняк

Житомир, 2018

УДК 004.65

Рекомендовано до друку Навчально-методичною радою Житомирського державного технологічного університету (протокол № 4 від 31.05.2018 р.)

Рецензенти:
Доцент кафедри КІ та КБ Житомирського державного технологічного університету, к.т.н., доцент Лобанчикова Н.М.

Декан факультету інформаційно-комп'ютерних технологій, к.т.н., доцент Морозов А.В.

Сугоняк І.І. Методичні вказівки до виконання курсових робіт з дисципліни "Бази даних"/ Єфіменко А.А., Ковальчук В.Н., Сугоняк І.І. - Житомир: ЖДТУ, 2018. – 112 с.

В методичній розробці наведено рекомендації щодо виконання лабораторних робіт, теоретичні відомості, контрольні питання, завдання та приклади виконання робіт. Методичні рекомендації призначені для студентів, що навчаються за напрямком 123 «Комп'ютерна інженерія» денної та заочної форм навчання.

УДК 004.65

ЗМІСТ

Лабораторна робота № 1

Дослідження серверів та реверс-інженерія баз даних

Мета роботи: дослідження системного каталогу та його представлень для отримання інформації про існуючі сервери та бази даних MSSQLServer.

Обладнання та інструментарій: ПК, MS SQL Server, MS Management Studio(Visual Studio)

Завдання на лабораторну роботу

Виконати дослідження інформаційної інфраструктури за наступним планом:

СЕРВЕРА
1. Інформація про сервера.
2. Список всіх баз даних

БАЗИ ДАНИХ
3. Розміщення файлів баз даних

ТАБЛИЦІ
4. Кількість записів в таблиці
5. Пошук кластерних індексів і куч (таблиць без кластерних індексів)
6. Активність в таблиці

ПРЕДСТАВЛЕННЯ

ПРОЦЕДУРИ, ЩО ЗБЕРІГАЮТЬСЯ

ФУНКЦІЇ
7. Стовпці
8. Значення по замовченю
9. Обчислювальні стовпці
10. Стовпці identity
11. Ключі і індекси
12. Зовнішні ключі
13. Залежності

Хід роботи

СЕРВЕРА

Інформація про сервера.

```
-- Іменасервераіекземпляра
Select @@SERVERNAME as [Server\Instance];
-- версія SQL Server
Select @@VERSION as SQLServerVersion;
-- екземпляр SQL Server
Select @@ServiceName AS ServiceInstance;
-- Поточна БД (БД, в контексті якої виконується запит)
Select DB_NAME() AS CurrentDB_Name;
```

Термін работи SQL Server після останього перезапуску

Системная база данных tempdb пересоздаётся при каждом перезапуске SQL Server. Вот один из способов определения времени последнего перезапуска сервера.

```
SELECT @@Servername AS ServerName ,
create_date AS  ServerStarted ,
DATEDIFF(s, create_date, GETDATE()) / 86400.0AS DaysRunning ,
DATEDIFF(s, create_date, GETDATE()) AS SecondsRunnig
FROMsys.databases
WHERE   name = 'tempdb';
GO
```

Список всіх баз даних

На будь-якому сервері є чотири або п'ять системних баз даних (master, model, msdb, tempdb і distribution (для реплікації)). Ви, ймовірно, захочете виключити ці бази в наступних запитах. Дуже просто побачити список баз даних в SSMS, але, ці запити будуть нашими «будівельними блоками» для більш складних запитів.

Є кілька шляхів для отримання списку всіх БД на T-SQL і нижче ви побачите деякі з них. Кожен метод повертає схожий результат, але з деякими відмінності.

```
EXEC sp_helpdb;
--OR
EXEC sp_Databases;
--OR
SELECT  @@SERVERNAME AS Server ,
        name ASDBName ,
        recovery_model_Desc ASRecoveryModel ,
        Compatibility_level ASCompatiblityLevel ,
        create_date ,
        state_desc
FROMsys.databases
ORDERBY Name;
--OR
SELECT  @@SERVERNAME AS Server ,
        d.name ASDBName ,
        create_date ,
        compatibility_level ,
        m.physical_name AS FileName
FROMsys.databases d
JOINsys.master_files m ON d.database_id = m.database_id
WHERE   m.[type] = 0 -- data files only
ORDERBY d.name;
GO
```

БАЗИ ДАНИХ

Можна зібрати інформацію про об'єкти у всіх БД, використовуючи різні уявлення каталогу і dmv. Більшість із запитів, поданих в цьому розділі, дивляться «всередину» тільки однієї БД, тому не забувайте вибирати потрібну БД в SSMS або за допомогою команди use database. Також пам'ятайте, що ви завжди можете подивитися в контексті якої БД буде виконаний запит, за допомогою select db_name ().
Системна таблиця sys.objects одна з ключових для збору інформації про об'єкти, що становлять вашу модель даних.

```
-- В прикладі в WHERE U - таблиці
USE MyDatabase;
GO
SELECT  *
FROMsys.objects
WHERE   type = 'U';
```

Завдання. Перелік і розшифровку значень type оформити у вигляді таблиці, дивіться документацію на sys.objects в MSDNsys.objects.type Інші подання каталогу, такі як sys.tables і sys.views, звертаються до sys.objects і надають інформацію про конкретному типі об'єктів. З цими уявленнями, плюс функцією OBJECTPROPERTY, ми можемо отримати величезну кількість інформації по кожному з об'єктів, що становлять нашу схему БД.

Розміщення файлів баз даних

Фізичне розташування обраної БД, включаючи основний файл даних (mdf), і файл журналу транзакцій (ldf), можуть бути отримані за допомогою цих запитів.

```
EXEC sp_Helpfile;
--OR
SELECT  @@Servername AS Server ,
        DB_NAME() AS DB_Name ,
        File_id ,
        Type_desc ,
Name ,
LEFT(Physical_Name, 1) AS Drive ,
        Physical_Name ,
RIGHT(physical_name, 3) AS Ext ,
Size ,
        Growth
FROMsys.database_files
ORDERBY File_id;
GO
```

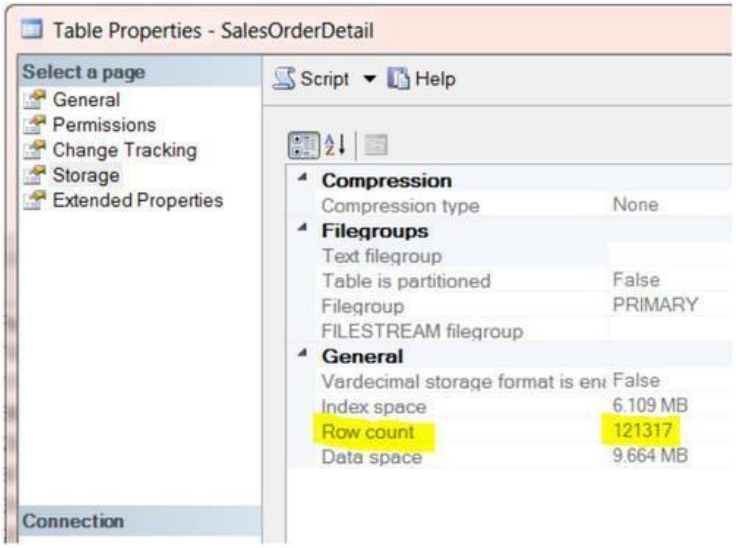
ТАБЛИЦІ

Звичайно, Object Explorer в SSMS показує повний список таблиць в обраній БД, але частина інформації за допомогою GUI отримати складніше, ніж за допомогою скриптів. Стандарт ANSI передбачає звернення до уявленням INFORMATION_SCHEMA, але вони не нададуть інформацію про об'єкти, які не є частиною стандарту (такі як тригери, extended procedures і т.д.), тому краще використовувати подання каталогу SQL Server.

```
EXEC sp_tables; -- Пам'ятайте, що цей метод поверне і таблиці, і
представления
--OR
SELECT  @@Servername AS ServerName ,
        TABLE_CATALOG ,
        TABLE_SCHEMA ,
        TABLE_NAME
FROM    INFORMATION_SCHEMA.TABLES
WHERE   TABLE_TYPE = 'BASE TABLE'
ORDERBY TABLE_NAME ;
--OR
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
        o.name AS'TableName' ,
        o.[Type] ,
o.create_date
FROMsys.objects o
WHEREo.Type = 'U' -- Usertable
ORDERBY o.name;
--OR
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
t.NameAS TableName,
        t.[Type],
t.create_date
FROMsys.tables t
ORDERBY t.Name;
GO
```

Кількість записів у таблиці

Якщо ви нічого не знаєте про таблиці, то все таблиці однаково важливі. Чим більше ви дізнаєтеся про таблиці, тим більше ви їх поділяєте на умовно важливіші і умовно менш важливі. В цілому, таблиці з величезною кількістю записів частіше роблять серйозний вплив на продуктивність.У SSMS ми можемо натиснути правою кнопкою миші на будь-яку таблицю, відкрити властивості на вкладці Storage і побачити кількість записів в таблиці.



Досить важко зібрати вручну цю інформацію про всі таблицях. Знову ж таки, якщо ми будемо писати SELECT COUNT (*) FROM TABLENAME для кожної таблиці, нам доведеться дуже багато друкувати.
Набагато зручніше використовувати T-SQL для генерування скрипта. Скрипт, наведений нижче, згенерує набір інструкцій T-SQL для отримання кількості рядків в кожній таблиці поточної бази даних. Просто виконайте його, скопіюйте результат в нове вікно і запустіть.

```
SELECT'Select '" + DB_NAME() + '.' + SCHEMA_NAME(SCHEMA_ID) + '.'
      + LEFT(o.name, 128) + '" as DBName, count(*) as Count From ' +
SCHEMA_NAME(SCHEMA_ID) + '.' + o.name
      + ';'AS' Script generator to get counts for all tables'
FROMsys.objects o
WHERE   o.[type] = 'U'
ORDERBY o.name;
```

Sp_msforeachtable - це недокументированная функція, яка «проходить» по всіх таблиць в БД і виконує запит, підставляючи замість '?' Ім'я поточної таблиці. Так само існує схожа функція sp_msforeachdb, що працює на рівні баз даних.Відомо кілька проблем з цієї недокументованою функцією, наприклад, використання спецсимволов в іменах об'єктів. Тобто якщо ім'я таблиці або бази даних містить знак '-', збережена процедура, лістинг якої нижче, завершиться з помилкою.

```
CREATETABLE #rowcount
( TablenameVARCHAR(128) ,
    Rowcnt INT );
EXEC sp_MSforeachtable ' insertinto #rowcount select"?", count(*) from ?'
SELECT  *
FROM    #rowcount
ORDER BY Tablename ,
```

```
Rowcnt;
DROP TABLE #rowcount;
```

Пошук кластер них індексів куп (таблиць без кластерних індексів)

Найшвидший спосіб отримання кількості записів в таблиці - отримувати кількість записів в кластерному індексі або купі. Пам'ятайте, що хоч цей метод і найшвидший, MS каже, що інформація про кількість записів індексу і реальна кількість рядків в таблиці може не збігатися, через те, що на оновлення інформації потрібно хоч і невелике, але час. У більшості ж випадків, ці значення або однакові, або дуже-дуже близькі і незабаром стануть однаковими.

```
-- Найшвидший шлях отримання кількості записів
- Hint: отримуйте з індексу, а не таблиці
SELECT  @@ServerName AS Server ,
        DB_NAME() AS DBName ,
        OBJECT_SCHEMA_NAME(p.object_id) AS SchemaName ,
        OBJECT_NAME(p.object_id) AS TableName ,
        i.Type_Desc ,
        i.Name ASIndexUsedForCounts ,
SUM(p.Rows) ASRows
FROMsys.partitions p
JOINsys.indexes i ON i.object_id = p.object_id
ANDi.index_id = p.index_id
WHEREi.type_desc IN ( 'CLUSTERED', 'HEAP' )
-- This iskey (1 index per table)
AND OBJECT_SCHEMA_NAME(p.object_id) <>'sys'
GROUPBYp.object_id ,
        i.type_desc ,
        i.Name
ORDERBYSchemaName ,
TableName;
-- OR
-- Схожий метод отримання кількості записів, але з використанням DMV
dm_db_partition_stats
SELECT  @@ServerName AS ServerName ,
        DB_NAME() AS DBName ,
        OBJECT_SCHEMA_NAME(ddps.object_id) AS SchemaName ,
        OBJECT_NAME(ddps.object_id) AS TableName ,
        i.Type_Desc ,
        i.Name ASIndexUsedForCounts ,
SUM(ddps.row_count) ASRows
FROM    sys.dm_db_partition_stats ddps
JOINsys.indexes i ON i.object_id = ddps.object_id
ANDi.index_id = ddps.index_id
```

```
WHEREi.type_desc IN ( 'CLUSTERED', 'HEAP' )
-- This iskey (1 index per table)
AND OBJECT_SCHEMA_NAME(ddps.object_id) <>'sys'
GROUPBYddps.object_id ,
        i.type_desc ,
        i.Name
ORDERBYSchemaName ,
TableName;
GO
```

Робота з кучами - це як робота з плоским файлом, замість бази даних. Якщо ви хочете гарантовано отримувати повне сканування таблиці при виконанні будь-якого запиту, використовуйте купи. Зазвичай я рекомендую додавати primary key до всіх таблиць-купах.

```
-- Купи (метод 1)
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
t.NameAS HeapTable ,
t.Create_Date
FROMsys.tables t
INNERJOINsys.indexes i ON t.object_id = i.object_id
ANDi.type_desc = 'HEAP'
ORDERBYt.Name
--OR
-- Купи (Метод2)
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
t.NameAS HeapTable ,
t.Create_Date
FROMsys.tables t
WHEREOBJECTPROPERTY(OBJECT_ID, 'TableHasClustIndex') = 0
ORDERBY t.Name;
--OR
-- Кучи (Метод 3) + количество записей
SELECT  @@ServerName AS Server ,
        DB_NAME() AS DBName ,
        OBJECT_SCHEMA_NAME(ddps.object_id) AS SchemaName ,
        OBJECT_NAME(ddps.object_id) AS TableName ,
        i.Type_Desc ,
SUM(ddps.row_count) ASRows
FROM    sys.dm_db_partition_stats AS ddps
JOINsys.indexes i ON i.object_id = ddps.object_id
ANDi.index_id = ddps.index_id
WHEREi.type_desc = 'HEAP'
```

```
AND OBJECT_SCHEMA_NAME(ddps.object_id) <>'sys'
GROUPBYddps.object_id ,
      i.type_desc
ORDERBY TableName;
```

Активність в таблиці

При роботах по оптимізації продуктивності, дуже важливо знати які таблиці активно читаються, а в які йде активна запис. Раніше ми дізналися скільки записів в наших таблицях, зараз подивимося як часто в них пишуть і читають. Пам'ятайте, що ця інформація з dmv, очищається при кожному перезапуску SQL Server. Чим довше сервер працює, тим надійніша статистика. Я відчуваю себе набагато впевненіше зі статистикою, зібраною за 30 днів, ніж зі статистикою, зібраною за тиждень.

```
-- Читання / запис таблиці
-- Купи не розглядаються, у них немає індексів
-- Тільки ті таблиці, до яких зверталися після запуску SQL Server
SELECT  @@ServerName AS ServerName ,
      DB_NAME() AS DBName ,
      OBJECT_NAME(ddius.object_id) AS TableName ,
SUM(ddius.user_seeks + ddius.user_scans + ddius.user_lookups)
AS  Reads ,
SUM(ddius.user_updates) ASWrites ,
SUM(ddius.user_seeks + ddius.user_scans + ddius.user_lookups
          + ddius.user_updates) AS [Reads&Writes] ,
( SELECT    DATEDIFF(s, create_date, GETDATE()) / 86400.0
FROMmaster.sys.databases
WHERE     name = 'tempdb'
) ASSampleDays ,
( SELECT    DATEDIFF(s, create_date, GETDATE()) AS SecoundsRunnig
FROMmaster.sys.databases
WHERE     name = 'tempdb'
      ) AS SampleSeconds
FROM    sys.dm_db_index_usage_stats ddius
INNERJOINsys.indexes i ON ddius.object_id = i.object_id
ANDi.index_id = ddius.index_id
WHEREOBJECTPROPERTY(ddius.object_id, 'IsUserTable') = 1
ANDddius.database_id = DB_ID()
GROUPBY OBJECT_NAME(ddius.object_id)
ORDERBY [Reads&Writes] DESC;
GO
```

Набагато більш просунута версія цього запиту представлена курсором, яке збирає інформацію по всіх таблиць всіх баз даних на сервері. Курсори мають

невисоку продуктивність, але переміщення по різних базах даних - це відмінне застосування для них.

```
--Операції читання і запису
--Купи пропущені, у них немає індексів
--Тільки таблиці, що використовувалися після перезапуску SQL Server
--У запиті використовується курсор для отримання інформації в усіх БД
--Єдиний звіт, зберігається в tempdb
DECLARE DBNameCursor CURSOR
FOR
SELECT  Name
FROMsys.databases
WHERE    Name NOTIN( 'master', 'model', 'msdb', 'tempdb',
'distribution' )
ORDERBY Name;
DECLARE @DBName NVARCHAR(128)
DECLARE @cmd VARCHAR(4000)
IF OBJECT_ID(N'tempdb..TempResults') IS NOT NULL
BEGIN
DROPTABLEtempdb..TempResults
END
CREATETABLEtempdb..TempResults
   (
     ServerName NVARCHAR(128) ,
     DBName NVARCHAR(128) ,
     TableName NVARCHAR(128) ,
     Reads INT ,
     Writes INT ,
     ReadsWrites INT ,
     SampleDays DECIMAL(18, 8) ,
     SampleSeconds INT
   )
OPEN DBNameCursor
FETCHNEXTFROM DBNameCursor INTO @DBName
WHILE @@fetch_status = 0
BEGIN
-------------------------------------------
-- Print @DBName
SELECT   @cmd = 'Use ' + @DBName + '; '
SELECT   @cmd = @cmd + ' Insert Into tempdb..TempResults
SELECT @@ServerName AS ServerName,
DB_NAME() AS DBName,
object_name(ddius.object_id) AS TableName ,
SUM(ddius.user_seeks
```

```
+ ddius.user_scans
+ ddius.user_lookups) AS Reads,
SUM(ddius.user_updates) as Writes,
SUM(ddius.user_seeks
+ ddius.user_scans
+ ddius.user_lookups
+ ddius.user_updates) as ReadsWrites,
(SELECT datediff(s,create_date, GETDATE()) / 86400.0
FROM sys.databases WHERE name = "tempdb") AS SampleDays,
(SELECT datediff(s,create_date, GETDATE())
FROM sys.databases WHERE name = "tempdb") as SampleSeconds
FROM sys.dm_db_index_usage_stats ddius
INNER JOIN sys.indexes i
ON ddius.object_id = i.object_id
AND i.index_id = ddius.index_id
WHERE objectproperty(ddius.object_id,"IsUserTable") = 1 --True
AND ddius.database_id = db_id()
GROUP BY object_name(ddius.object_id)
ORDER BY ReadsWrites DESC;'
--PRINT @cmd
EXECUTE (@cmd)
```

```
FETCHNEXTFROM DBNameCursor INTO @DBName
END
CLOSE DBNameCursor
DEALLOCATE DBNameCursor
SELECT  *
FROMtempdb..TempResults
ORDERBYDBName ,
      TableName;
--DROP TABLE tempdb..TempResults;
```

ПРЕДСТАВЛЕННЯ

У SQL Server, в деяких випадках, ми можемо оновлювати дані з використанням уявлення. Щоб отримати уявлення «тільки для читання», можна використовувати SELECT DISTINCT при його створенні. Дані «через» уявлення можна змінювати тільки в тому випадку, якщо кожному рядку уявлення відповідає тільки один рядок в «базової» таблиці. Будь-яке уявлення, що не відповідає цьому критерію, тобто побудоване на кількох таблицях, або з використанням угруповань, агрегатних функцій і обчислень, буде доступно тільки для читання.

```
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DBName ,
```

```
      o.name ASViewName ,
      o.[Type] ,
o.create_date
FROMsys.objects o
WHERE   o.[Type] = 'V' -- View
ORDERBY o.NAME
--OR
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DBName ,
      Name ASViewName ,
      create_date
FROMsys.Views
ORDERBY Name
--OR
SELECT  @@Servername AS ServerName ,
      TABLE_CATALOG ,
      TABLE_SCHEMA ,
      TABLE_NAME ,
      TABLE_TYPE
FROM    INFORMATION_SCHEMA.TABLES
WHERE   TABLE_TYPE  =  'VIEW'
ORDERBY TABLE_NAME
--OR
-- CREATEVIEW Code
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DB_Name ,
      o.name AS'ViewName' ,
o.Type ,
o.create_date ,
      sm.[DEFINITION] AS'View script'
FROMsys.objects o
INNERJOIN sys.sql_modules sm ONo.object_id = sm.OBJECT_ID
WHEREo.Type = 'V' -- View
ORDERBY o.NAME;
GO
```

ЗБЕРЕЖЕНІ ПРОЦЕДУРИ

Збережені процедури - це група скриптів, які компілюються в єдиний план виконання. Ми можемо використовувати подання каталогу, щоб визначити які ХП створені, які дії вони виконують і над якими таблицями.

```
-- Збережені процедури
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DBName ,
```

```
        o.name ASStoredProcedureName ,
        o.[Type] ,
o.create_date
FROMsys.objects o
WHERE   o.[Type] = 'P' -- Stored Procedures
ORDERBY o.name
--OR
-- Дополнительная информация о ХП
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DB_Name ,
        o.name AS'ViewName' ,
        o.[type] ,
o.Create_date ,
        sm.[definition] AS'Stored Procedure script'
FROMsys.objects o
INNERJOIN sys.sql_modules sm ONo.object_id = sm.object_id
WHERE   o.[type] = 'P' -- Stored Procedures
        -- AND sm.[definition] LIKE'%insert%'
        -- AND sm.[definition] LIKE'%update%'
        -- AND sm.[definition] LIKE'%delete%'
        -- AND sm.[definition] LIKE'%tablename%'
ORDERBY o.name;
GO
```

Додавши просте умова в WHERE ми можемо отримати інформацію тільки про тих процедурах, які, наприклад, виконують операції INSERT.

```
WHERE   o.[type]  = 'P' -- Stored Procedures
       AND sm.definition LIKE '%insert%'
ORDER BY o.name
…
```

Трохи модифікувавши умова в WHERE, ми можемо зібрати інформацію про ХП, які виробляють оновлення, видалення або ж звертаються до певних таблиць.

ФУНКЦІЇ

Функції зберігаються в SQL Server, приймають будь-які параметри і виконують певні дії, або обчислення, після чого повертають результат.

```
-- Функції
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DB_Name ,
        o.name AS'Functions' ,
        o.[Type] ,
o.create_date
FROMsys.objects o
WHEREo.Type = 'FN' -- Function
```

```
ORDERBY o.NAME;
--OR
-- Дополнительная информация о функциях
SELECT @@Servername AS ServerName ,
DB_NAME() AS DB_Name ,
       o.name AS'FunctionName' ,
       o.[type] ,
o.create_date ,
       sm.[DEFINITION] AS'Function script'
FROMsys.objects o
INNERJOIN sys.sql_modules sm ONo.object_id = sm.OBJECT_ID
WHERE   o.[Type] = 'FN' -- Function
ORDERBY o.NAME;
GO
```

ТРИГЕРИ

Тригер - це щось на зразок збереженої процедури, яка виконується у відповідь на певні дії з тієї таблицею, яку цей тригер належить. Наприклад, ми можемо створити INSERT, UPDATE і DELETE тригери.

```
-- Тригери
SELECT @@Servername AS ServerName ,
DB_NAME() AS DBName ,
       parent.name ASTableName ,
       o.name ASTriggerName ,
       o.[Type] ,
o.create_date
FROMsys.objects o
INNERJOINsys.objects parent ON o.parent_object_id = parent.object_id
WHEREo.Type = 'TR' -- Triggers
ORDERBYparent.name ,
       o.NAME
--OR
SELECT  @@Servername AS ServerName ,
       DB_NAME() AS DB_Name ,
       Parent_id ,
       name ASTriggerName ,
       create_date
FROMsys.triggers
WHERE   parent_class = 1
ORDERBY name;
--OR
-- Дополнительная информация о триггерах
SELECT  @@Servername AS ServerName ,
       DB_NAME() AS DB_Name ,
       OBJECT_NAME(Parent_object_id) ASTableName ,
```

```
      o.name AS'TriggerName' ,
o.Type ,
o.create_date ,
      sm.[DEFINITION] AS'Trigger script'
FROMsys.objects o
INNERJOIN sys.sql_modules sm ONo.object_id = sm.OBJECT_ID
WHEREo.Type = 'TR' -- Triggers
ORDERBY o.NAME;
GO
```

CHECK-обмеження

CHECK-обмеження - це непоганий засіб для реалізації бізнес-логіки в базі даних. Наприклад, деякі поля повинні бути позитивними, або негативними, або дата в одному стовпці повинна бути більше дати в іншому.

```
-- Check Constraints
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DBName ,
      parent.name AS'TableName' ,
      o.name AS'Constraints' ,
      o.[Type] ,
o.create_date
FROMsys.objects o
INNERJOINsys.objects parent
ONo.parent_object_id = parent.object_id
WHEREo.Type = 'C' -- CheckConstraints
ORDERBYparent.name ,
      o.name
--OR
--CHECK constriant definitions
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DBName ,
      OBJECT_SCHEMA_NAME(parent_object_id) ASSchemaName ,
      OBJECT_NAME(parent_object_id) ASTableName ,
      parent_column_id AS  Column_NBR ,
      Name AS CheckConstraintName ,
type ,
      type_desc ,
      create_date ,
      OBJECT_DEFINITION(object_id) AS CheckConstraintDefinition
FROMsys.Check_constraints
ORDERBYTableName ,
SchemaName ,
      Column_NBR
GO
```

МОДЕЛЬ ДАНИХ

Стовпці

Наступний скрипт описує таблиці і стовпчики з усієї бази даних. Результат цього запиту, можна скопіювати в Excel, де можна налаштувати фільтри і сортування і добре розібратися з типами даних, що використовуються в БД. Так само, зверніть увагу на стовпці з однаковими іменами, але різними типами даних.

```
SELECT  @@Servername AS Server ,
        DB_NAME() AS DBName ,
isc.Table_Name AS TableName ,
isc.Table_Schema AS SchemaName ,
        Ordinal_Position AS  Ord ,
        Column_Name ,
        Data_Type ,
        Numeric_Precision AS  Prec ,
        Numeric_Scale AS  Scale ,
        Character_Maximum_Length ASLEN , -- -1 means MAXlikeVarchar(MAX)
        Is_Nullable ,
        Column_Default ,
        Table_Type
FROM    INFORMATION_SCHEMA.COLUMNS isc
INNERJOIN information_schema.tables ist
ONisc.table_name = ist.table_name
--      WHERE Table_Type = 'BASE TABLE' -- 'Base Table'or'View'
ORDERBYDBName ,
TableName ,
SchemaName ,
Ordinal_position;
--Імена стовпців і кількість повторів
--Використовується для пошуку однойменних стовпців з різними типами даних / довжиною
SELECT  @@Servername AS Server ,
        DB_NAME() AS DBName ,
        Column_Name ,
        Data_Type ,
        Numeric_Precision AS  Prec ,
        Numeric_Scale AS  Scale ,
        Character_Maximum_Length ,
COUNT(*) ASCount
FROM    information_schema.columns isc
INNERJOIN information_schema.tables ist
ONisc.table_name = ist.table_name
WHERE   Table_type = 'BASE TABLE'
```

```
GROUPBY Column_Name ,
        Data_Type ,
        Numeric_Precision ,
        Numeric_Scale ,
Character_Maximum_Length;
-- Информация по используемым типам данных
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
        Data_Type ,
        Numeric_Precision AS  Prec ,
        Numeric_Scale AS  Scale ,
        Character_Maximum_Length AS [Length] ,
COUNT(*) ASCOUNT
FROM     information_schema.columns isc
INNERJOIN information_schema.tables ist
ONisc.table_name = ist.table_name
WHERE   Table_type = 'BASE TABLE'
GROUPBY Data_Type ,
        Numeric_Precision ,
        Numeric_Scale ,
        Character_Maximum_Length
ORDERBY Data_Type ,
        Numeric_Precision ,
        Numeric_Scale ,
        Character_Maximum_Length
-- Large object data types or Binary Large Objects(BLOBs)
-- Пам'ятайте, що індекси по цих таблиць не можуть бути перебудовані в
режимі "online"
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
isc.Table_Name ,
        Ordinal_Position AS  Ord ,
        Column_Name ,
        Data_Type AS  BLOB_Data_Type ,
        Numeric_Precision AS  Prec ,
        Numeric_Scale AS  Scale ,
        Character_Maximum_Length AS [Length]
FROM     information_schema.columns isc
INNERJOIN information_schema.tables ist
ONisc.table_name = ist.table_name
WHERE   Table_type = 'BASE TABLE'
AND( Data_Type IN ( 'text', 'ntext', 'image', 'XML' )
OR( Data_Type IN ( 'varchar', 'nvarchar', 'varbinary' )
AND Character_Maximum_Length = -1
                )
```

```
) -- varchar(max), nvarchar(max), varbinary(max)
ORDERBYisc.Table_Name ,
Ordinal_position;
```

Значения за замовченям

Значення за замовчуванням - це значення, яке буде збережено, якщо ніякого значення для стовпців буде задано при вставці. Найчастіше, для стовпців зберігають дату ставлять get_date (). Також, значення за замовчуванням використовуються для аудиту - вставляється system_user для визначення облікового запису користувача, який здійснив певну дію.

```
-- Table Defaults
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
        parent.name ASTableName ,
        o.name ASDefaults ,
        o.[Type] ,
o.Create_date
FROMsys.objects o
INNERJOINsys.objects parent
ONo.parent_object_id = parent.object_id
WHERE   o.[Type] = 'D' -- Defaults
ORDERBYparent.name ,
        o.NAME
--OR
-- Column Defaults
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DB_Name ,
        OBJECT_SCHEMA_NAME(parent_object_id) ASSchemaName ,
        OBJECT_NAME(parent_object_id) ASTableName ,
        parent_column_id AS  Column_NBR ,
        Name ASDefaultName ,
        [type] ,
        type_desc ,
        create_date ,
        OBJECT_DEFINITION(object_id) AS Defaults
FROM    sys.default_constraints
ORDERBYTableName ,
        Column_NBR
--OR
-- Column Defaults
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DB_Name ,
        OBJECT_SCHEMA_NAME(t.object_id) AS SchemaName ,
```

```
t.NameAS TableName ,
        c.Column_ID ASOrd ,
        c.Name AS Column_Name ,
        OBJECT_NAME(default_object_id) ASDefaultName ,
        OBJECT_DEFINITION(default_object_id) AS Defaults
FROMsys.Tables t
INNERJOINsys.columns c ON t.object_id = c.object_id
WHERE    default_object_id <>0
ORDERBYTableName ,
SchemaName ,
        c.Column_ID
GO
```

Обчислювальні стовпці

Обчислювані стовпці - це стовпці, значення в яких обчислюються на підставі, як правило, значний в інших стовпцях таблиці.

```
-- Обчислювані стовпці
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
        OBJECT_SCHEMA_NAME(object_id) ASSchemaName ,
        OBJECT_NAME(object_id) ASTablename ,
        Column_id ,
        Name AS  Computed_Column ,
        [Definition] ,
        is_persisted
FROMsys.computed_columns
ORDERBYSchemaName ,
Tablename ,
        [Definition];
--Or
-- Computed Columns
SELECT  @@Servername AS ServerName ,
        DB_NAME() AS DBName ,
        OBJECT_SCHEMA_NAME(t.object_id) AS SchemaName,
t.NameAS TableName ,
        c.Column_ID ASOrd ,
        c.Name AS Computed_Column
FROMsys.Tables t
INNERJOINsys.Columns c ON t.object_id = c.object_id
WHERE   is_computed = 1
ORDERBYt.Name ,
SchemaName ,
        c.Column_ID
GO
```

Стовпці identity

Стовпці IDENTITY автоматично заповнюються системою унікальними значеннями. Зазвичай використовуються для зберігання порядкового номера запису в таблиці.

```
SELECT @@Servername AS ServerName ,
DB_NAME() AS DBName ,
       OBJECT_SCHEMA_NAME(object_id) ASSchemaName ,
       OBJECT_NAME(object_id) ASTableName ,
       Column_id ,
       Name AS  IdentityColumn ,
       Seed_Value ,
       Last_Value
FROMsys.identity_columns
ORDERBYSchemaName ,
TableName ,
       Column_id;
GO
```

Ключі і індекси

Наявність первинного ключа та відповідного індексу у таблиці - це одна з best practice. Ще одна best practice полягає в тому, що зовнішні ключі так само повинні мати індекс, побудований за стовпцями, що входять у зовнішній ключ. Індекси, побудовані «за зовнішніми ключам» відмінно підходять для з'єднання таблиць. Ці індекси так само добре позначаються на продуктивності при видаленні записів.

Скрипт для пошуку всіх індексів у всіх таблицях поточної БД.

```
SELECT  @@Servername AS ServerName ,
       DB_NAME() AS DB_Name ,
o.NameAS TableName ,
       i.Name AS IndexName
FROMsys.objects o
INNERJOINsys.indexes i ON o.object_id = i.object_id
WHEREo.Type = 'U' -- Usertable
ANDLEFT(i.Name, 1) <>'_' -- Remove hypothetical indexes
ORDERBY o.NAME ,
       i.name;
GO
```

На підставі раніше виконувалися запитів, SQL Server надає інформацію про відсутніх індексах в БД, створення яких може збільшити продуктивність.Не додавайте ці індекси наосліп. Використання включених стовпців, наприклад, може відгукнутися серйозним збільшенням обсягів.

```
-- Отсутствующие индексы из DMV
SELECT  @@ServerName AS ServerName ,
        DB_NAME() AS DBName ,
        t.name AS'Affected_table' ,
( LEN(ISNULL(ddmid.equality_columns, N'')
+ CASEWHENddmid.equality_columns ISNOTNULL
ANDddmid.inequality_columns  ISNOTNULLTHEN','
ELSE''
END) - LEN(REPLACE(ISNULL(ddmid.equality_columns, N'')
+ CASEWHENddmid.equality_columns
ISNOTNULL
ANDddmid.inequality_columns
ISNOTNULL
THEN','
ELSE''
END, ',', '')) ) + 1AS K ,
COALESCE(ddmid.equality_columns, '')
        + CASEWHENddmid.equality_columns ISNOTNULL
ANDddmid.inequality_columns ISNOTNULLTHEN','
ELSE''
END + COALESCE(ddmid.inequality_columns, '') AS Keys ,
COALESCE(ddmid.included_columns, '') AS [include] ,
'Create NonClustered Index IX_' + t.name + '_missing_'
        + CAST(ddmid.index_handle ASVARCHAR(20))
        + ' On ' + ddmid.[statement] COLLATE database_default
+ ' (' + ISNULL(ddmid.equality_columns, '')
+ CASEWHENddmid.equality_columns ISNOTNULL
ANDddmid.inequality_columns  ISNOTNULLTHEN','
ELSE''
END + ISNULL(ddmid.inequality_columns, '') + ')'
+ ISNULL(' Include (' + ddmid.included_columns + ');', ';')
AS sql_statement ,
        ddmigs.user_seeks ,
        ddmigs.user_scans ,
CAST(( ddmigs.user_seeks + ddmigs.user_scans )
        * ddmigs.avg_user_impact AS BIGINT) AS'est_impact' ,
avg_user_impact ,
ddmigs.last_user_seek ,
( SELECT    DATEDIFF(Second, create_date, GETDATE()) Seconds
FROMsys.databases
```

```
WHERE     name = 'tempdb'
      ) SecondsUptime
FROM    sys.dm_db_missing_index_groups ddmig
INNERJOIN sys.dm_db_missing_index_group_stats ddmigs
ONddmigs.group_handle = ddmig.index_group_handle
INNERJOIN sys.dm_db_missing_index_details ddmid
ONddmig.index_handle = ddmid.index_handle
INNERJOINsys.tables t ON ddmid.OBJECT_ID = t.OBJECT_ID
WHEREddmid.database_id = DB_ID()
ORDERBY est_impact DESC;
GO
```

Зовнішні ключі

Зовнішні ключі визначають зв'язок між таблицями і використовуються для контролю посилальної цілісності. На діаграмі сутність-зв'язок лінії між таблицями позначають зовнішні ключі.

```
-- Foreign Keys
SELECT  @@Servername AS ServerName ,
      DB_NAME() AS DB_Name ,
      parent.name AS'TableName' ,
      o.name AS'ForeignKey' ,
      o.[Type] ,
o.Create_date
FROMsys.objects o
INNERJOINsys.objects parent ON o.parent_object_id = parent.object_id
WHERE   o.[Type] = 'F' -- Foreign Keys
ORDERBYparent.name ,
      o.name
--OR
SELECT  f.name AS ForeignKey ,
      SCHEMA_NAME(f.SCHEMA_ID) AS SchemaName ,
      OBJECT_NAME(f.parent_object_id) AS TableName ,
      COL_NAME(fc.parent_object_id, fc.parent_column_id) AS ColumnName ,
      SCHEMA_NAME(o.SCHEMA_ID) ReferenceSchemaName ,
      OBJECT_NAME(f.referenced_object_id) AS ReferenceTableName ,
      COL_NAME(fc.referenced_object_id, fc.referenced_column_id)
AS ReferenceColumnName
FROMsys.foreign_keys AS f
INNERJOINsys.foreign_key_columns AS fc
ONf.OBJECT_ID = fc.constraint_object_id
INNERJOINsys.objectsAS o ON o.OBJECT_ID = fc.referenced_object_id
ORDERBYTableName ,
      ReferenceTableName;
```

```
GO
```

Пропущені індекси за зовнішніми ключам
Бажано мати індекс, побудований за стовпцями, що входять у зовнішній ключ. Це значно прискорює з'єднання таблиць, які, зазвичай, все одно з'єднуються по зовнішньому ключу. Ці індекси так само значно прискорюють операції видалення. Якщо такого індексу немає, SQL Server буде виробляти table scan пов'язаної таблиці, при кожному видаленні записи з «першої» таблиці.

```
--Foreign Keys missing indexes
-- Пам'ятайте, що цей скрипт працює тільки для створення індексів по однім стовпці
-- Зовнішні ключі, що складаються більш ніж з одного стовпчика, не відслідковуються
SELECT  DB_NAME() AS DBName ,
rc.Constraint_Name AS FK_Constraint ,
-- rc.Constraint_Catalog AS FK_Database,
-- rc.Constraint_Schema AS FKSch,
ccu.Table_Name AS FK_Table ,
ccu.Column_Name AS FK_Column ,
       ccu2.Table_Name ASParentTable ,
       ccu2.Column_Name ASParentColumn ,
       I.Name ASIndexName ,
CASEWHENI.NameISNULL
THEN'IF NOT EXISTS (SELECT * FROM sys.indexes
                           WHERE object_id = OBJECT_ID(N'"
+ RC.Constraint_Schema + '.' + ccu.Table_Name
+ '") AND name = N"IX_' + ccu.Table_Name + '_'
+ ccu.Column_Name + '") '
               + 'CREATE NONCLUSTERED INDEX IX_' + ccu.Table_Name + '_'
               + ccu.Column_Name + ' ON ' + rc.Constraint_Schema + '.'
+ ccu.Table_Name + '( ' + ccu.Column_Name
               + ' ASC ) WITH (PAD_INDEX = OFF,
STATISTICS_NORECOMPUTE = OFF,
                           SORT_IN_TEMPDB = ON, IGNORE_DUP_KEY = OFF,
DROP_EXISTING = OFF, ONLINE = ON);'
ELSE''
ENDASSQL
FROM     information_schema.referential_constraints RC
JOIN INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ccu
ONrc.CONSTRAINT_NAME = ccu.CONSTRAINT_NAME
JOIN INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ccu2
ONrc.UNIQUE_CONSTRAINT_NAME = ccu2.CONSTRAINT_NAME
LEFTJOINsys.columns c ON ccu.Column_Name = C.name
```

```
ANDccu.Table_Name = OBJECT_NAME(C.OBJECT_ID)
LEFTJOINsys.index_columns ic ON C.OBJECT_ID = IC.OBJECT_ID
ANDc.column_id = ic.column_id
AND index_column_id  =1
                                -- index found has the foreignkey
--  as the firstcolumn
LEFTJOINsys.indexes i ON IC.OBJECT_ID = i.OBJECT_ID
ANDic.index_Id = i.index_Id
WHERE   I.name ISNULL
ORDERBY FK_table ,
ParentTable ,
      ParentColumn;
GO
```

Залежності

Існує три різні способи для «реверс-інжинірингу» залежностей в БД. Перший методу - використовувати збережену процедуру sp_msdependecies. Другий - системні таблиці, пов'язані зі зовнішніми ключами. Третій метод - використовувати CTE (CommonTableExpression).

Sp_msdependencies - це недокументированная збережена процедура, яка може бути дуже корисна для розбору складних взаємозалежностей таблиць

```
EXEC sp_msdependencies '?' -- Displays Help
sp_MSobject_dependencies name = NULL, type = NULL, flags = 0x01fd
  name:  name or null (all objects of type)
  type:  type number (see below) or null
        if both null, get all objects in database
  flags is a bitmask of the following values:
        0x10000  = return multiple parent/child rows per object
        0x20000  = descending return order
        0x40000  = return children instead of parents
        0x80000 = Include input object in output result set
0x100000 = return only firstlevel (immediate) parents/children
0x200000 = return only DRI dependencies
power(2, object type number(s)) to return in results set:
0 (1        - 0x0001)     - UDF
1 (2        - 0x0002)     - system tables or MS-internal objects
2 (4        - 0x0004)     - view
3 (8        - 0x0008)     - usertable
4 (16       - 0x0010)     - procedure
5 (32       - 0x0020)     - log
6 (64       - 0x0040)     - default
```

7 (128 - 0x0080) - rule
8 (256 - 0x0100) - trigger
12 (1024 - 0x0400) - uddt
shortcuts:
29 (0x011c) - trig, view, usertable, procedure
448 (0x00c1) - rule, default, datatype
4606 (0x11fd) - all but systables/objects
4607 (0x11ff) – all

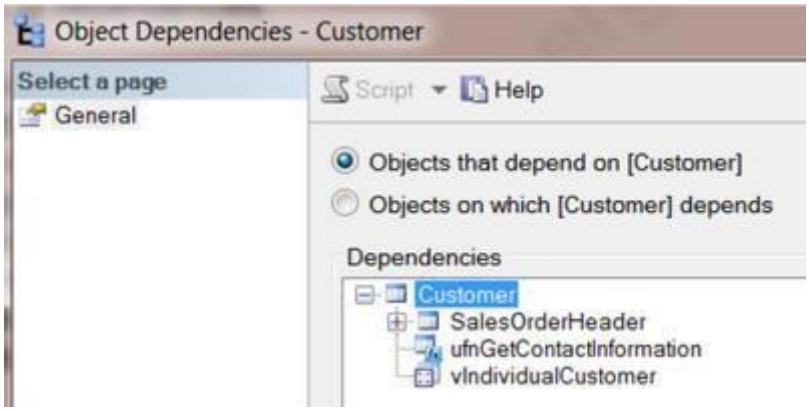
Якщо ми виведемо всі залежності, використовуючи sp_msdependencies, ми отримаємо чотири стовпці: Type, ObjName, Owner (Schema), Sequence. Зверніть увагу на номер послідовності (Sequence) - він починається з 1 і послідовно збільшується. Sequence - це «порядковий номер» залежності. Я кілька разів використовував цей метод, коли мені потрібно було виконати архівування або видалення на дуже великий БД. Якщо ви знаєте залежності таблиці, значить у вас є «дорожня карта» - в якому порядку вам потрібно архівувати або видаляти дані. Почніть з таблиці з найбільшим значення в стовпці Sequence і рухайтеся від нього в зворотному порядку - від більшого до меншого. Таблиці з однаковим значенням Sequence можуть бути видалені одночасно. Цей метод не порушує жодного з обмежень зовнішніх ключів і дозволяє перенести / видалити записи без тимчасового видалення і перестроювання обмежень (constraints).

```
EXEC sp_msdependencies NULL -- Все залежності в БД
EXEC sp_msdependencies NULL, 3 -- Залежно певної таблиці
```

Results | Messages

	oType	oObjName	oOwner	oSequence
2	8	AddressType	Person	1
3	8	AWBuildVersion	dbo	1
4	8	BusinessEntity	Person	1
5	8	ContactType	Person	1

У SSMS, якщо ви натиснете правою кнопкою миші на ім'я таблиці, ви зможете вибрати «View Dependencies» і «Об'єкти, які залежать від TABLENAME»:

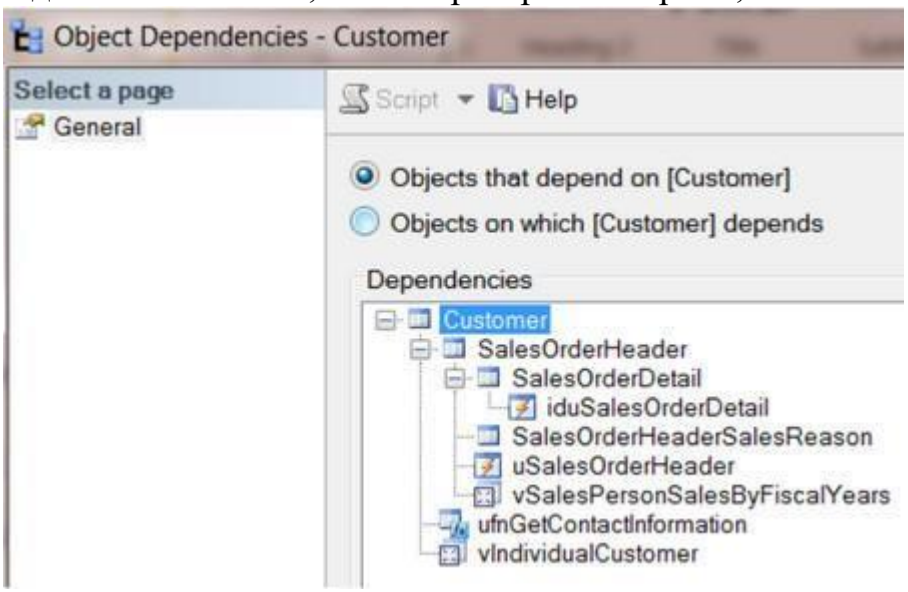


Ми також можемо отримати цю інформацію в такий спосіб:

```
-- sp_MSdependencies — Тільки верхній рівень:
-- Об'єкти, які залежать від зазначеного об'єкта
EXEC sp_msdependencies N'Sales.Customer',null, 1315327 -- Change Table Name
```

	oType	oObjName	oOwner	oSequence
1	1	ufnGetContactInformation	dbo	1
2	4	vIndividualCustomer	Sales	1
3	8	SalesOrderHeader	Sales	1
4	256	iduSalesOrderDetail	Sales	1

Якщо в SSMS, у вікні перегляду залежностей, вибрати «Об'єкти які залежать від TABLENAME», а потім розкрити всі рівні, ми побачимо наступне:

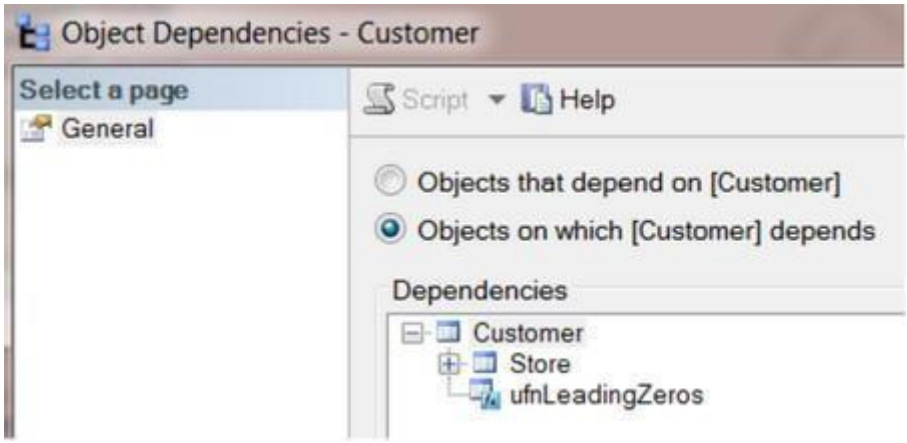


Ту ж саму інформацію поверне sp_msdependencies.

```
-- sp_MSdependencies - Всі рівні
-- Об'єкти, які залежать від зазначеного об'єкта
EXEC sp_MSdependencies N'Sales.Customer', NULL, 266751 -- Change Table Name
```

	oType	oObjName	oOwner	oSequence
1	1	ufnGetContactInformation	dbo	1
2	4	vIndividualCustomer	Sales	1
3	8	SalesOrderHeader	Sales	1
4	4	vSalesPersonSalesByFiscalYears	Sales	2
5	8	SalesOrderDetail	Sales	2
6	8	SalesOrderHeaderSalesReason	Sales	2
7	256	uSalesOrderHeader	Sales	2
8	256	iduSalesOrderDetail	Sales	3

Так само, в SSMS, ми можемо побачити від яких об'єктів залежить вибрана таблиця.



Наступний запит, з використанням msdependencies, поверне ту ж саму інформацію

```
-- Об'єкти, від яких залежить вказаний об'єкт
EXEC sp_MSdependencies N'Sales.Customer', null, 1053183 -- Change Table
```

	oType	oUDDTName	oOwner	oSequence
1	4096	Name	dbo	1
2	4096	NameStyle	dbo	1

	oType	oObjName	oOwner	oSequence
1	1	ufnLeadingZeros	dbo	2
2	8	Person	Person	2
3	8	SalesTerritory	Sales	2
4	8	Store	Sales	2

Якщо ви хочете отримати список залежностей таблиць, ви можете використовувати тимчасову таблицю, щоб відфільтрувати залежності за типом.

```
CREATETABLE #TempTable1
   (
     Type INT ,
     ObjName VARCHAR(256) ,
     Owner VARCHAR(25) ,
     Sequence INT
);
INSERTINTO #TempTable1
EXEC sp_MSdependencies NULL
SELECT  *
FROM    #TempTable1
WHERE   Type = 8 --Tables
ORDERBYSequence ,
ObjName
DROPTABLE #TempTable1;
```

Запити до системних уявленням каталогу
Другий метод «реверс-інжинірингу» залежностей у вашій БД - це запити до системних уявленням каталогу, пов'язаним зі зовнішніми ключами.

```
--Independent tables
SELECT NameAS InDependentTables
FROMsys.tables
WHERE   object_id NOTIN( SELECT referenced_object_id
FROMsys.foreign_key_columns )
-- Checkfor parents
AND object_id NOTIN( SELECT parent_object_id
FROMsys.foreign_key_columns )
-- Checkfor Dependents
ORDERBY Name
-- Tables with dependencies.
SELECTDISTINCT
      OBJECT_NAME(referenced_object_id) ASParentTable ,
      OBJECT_NAME(parent_object_id) ASDependentTable ,
      OBJECT_NAME(constraint_object_id) AS ForeignKeyName
FROMsys.foreign_key_columns
ORDERBYParentTable ,
      DependentTable
-- Top levelof the pyramid tables. Tables withno parents.
SELECTDISTINCT
      OBJECT_NAME(referenced_object_id) AS TablesWithNoParent
FROMsys.foreign_key_columns
WHERE    referenced_object_id NOTIN( SELECT parent_object_id
FROMsys.foreign_key_columns )
ORDERBY1
-- Bottom levelof the pyramid tables.
-- Tables withno dependents. (These are the leaves on a tree.)
SELECTDISTINCT
      OBJECT_NAME(parent_object_id) AS TablesWithNoDependents
FROMsys.foreign_key_columns
WHERE   parent_object_id NOTIN( SELECT referenced_object_id
FROMsys.foreign_key_columns )
ORDERBY1
-- Tables withboth parents and dependents.
-- Tables in the middle of the hierarchy
SELECTDISTINCT
      OBJECT_NAME(referenced_object_id) AS MiddleTables
FROMsys.foreign_key_columns
WHERE    referenced_object_id IN( SELECT parent_object_id
FROMsys.foreign_key_columns )
AND parent_object_id  NOTIN ( SELECT   referenced_object_id
```

```
FROMsys.foreign_key_columns )
ORDERBY1;
-- in rare cases, you might find a self-referencing dependent table.
-- Recursive (self) referencing table dependencies.
SELECTDISTINCT
      OBJECT_NAME(referenced_object_id) ASParentTable ,
      OBJECT_NAME(parent_object_id) ASChildTable ,
      OBJECT_NAME(constraint_object_id) AS ForeignKeyName
FROMsys.foreign_key_columns
WHERE    referenced_object_id = parent_object_id
ORDERBY1 ,
2;
```

Використання CTETретій метод, для отримання ієрархії залежностей - використання рекурсивного CTE.

```
-- How to find the hierarchical dependencies
-- Solve recursive queries using Common Table Expressions (CTE)
WITH     TableHierarchy (  ParentTable, DependentTable, Level )
AS (
-- Anchor member definition (First level group to start the process)
SELECTDISTINCT
CAST(NULLASINT) AS ParentTable ,
e.referenced_object_id AS DependentTable ,
0ASLevel
FROMsys.foreign_key_columns AS e
WHEREe.referenced_object_id NOTIN (
SELECT parent_object_id
FROMsys.foreign_key_columns )
-- Add filter dependents ofonly one parent table
-- AND Object_Name(e.referenced_object_id) = 'User'
UNIONALL
-- Recursive member definition (Find all the layers of dependents)
SELECT --Distinct
e.referenced_object_id AS ParentTable ,
e.parent_object_id AS DependentTable ,
Level + 1
FROMsys.foreign_key_columns AS e
INNERJOIN TableHierarchy AS d
ON( e.referenced_object_id ) =
d.DependentTable
          )
   -- Statement that executes the CTE
SELECTDISTINCT
```

```
        OBJECT_NAME(ParentTable) ASParentTable ,
        OBJECT_NAME(DependentTable) ASDependentTable ,
Level
FROM    TableHierarchy
ORDERBYLevel ,
ParentTable ,
        DependentTable;
```

Завдання на самостійну роботу

Провести аналогічне дослідження серверів та баз даних MySQL Server. Дослідити можливості команди SHOW.

Лабораторна робота № 2

Адміністрування баз данихSQL Server

Мета роботи: дослідження інструкцій управління доступом до баз та сховщ даних. Заходи резервування даних.

Обладнання та інструментарій: ПК, MS SQL Server, MS Management Studio(Visual Studio)

Завдання на лабораторну роботу

1. Управління доступом до БД
2. Резервування та відновлення БД

Приклади до опрацювання

Завдання 1. Управління доступом до БД

- Створити матрицю доступу до БД за метою курсової роботи.
- Для визначених користувачів (не менше 2 типів):
  - Створити обліковий запис та ім’я входу користувач SQLServer
  - Визначити для даного логіну користувача сервера БД з правами на створення нових баз даних
  - Перевірити наявність даного користувача
  - Для тестової БД даних створити користувача з правами на модифікацію схеми даних та виконання операцій вставки та перегляду таблиць.
  - Перевірити надані права доступу.

Завдання виконувати із використанням TSQL та системних процедур, що зберігаються.

Звіт містить 3 файли – файл БД, лістинг запитів та процедур створення користувачів, документ зі скріншотами екрану, що містять результати виконання завдань.

Створення матриці доступу

Найбільші права доступу до даних БД мають адміністратори. Для них необхідно призначити стандартні серверні ролі та ролі БД.

Для групи користувачів відносяться керівники підрозділів. Для цієї групи необхідно передбачити можливість передавати їх права та привілеї. Користувачі типу Wokers мають обмежений доступ до об’єктів БД. Більш детально перелік об’єктів БД, доступ до яких надано користувачам відділу (на прикладі відділу продаж) наведено в табл. 1.1. На перетині рядків і стовпців зазначено дії, які може виконувати користувач даного типу ролі: 0 – немає доступу, 1 – перегляд даних, 2 – редагування, 3 – видалення, 4 – вставка даних, 5 - повний доступ(виконання процедури).

Таблиця 1.1

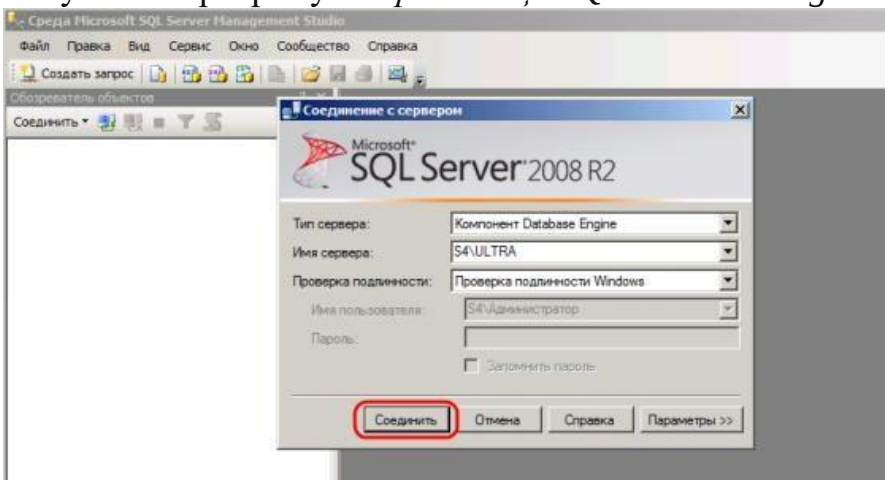
Матриця доступу для співробітників відділу продажу

	Керівник відділу	Менеджер	Комірник
Товари	5*	1	1,2,3
Замовлені товари	5*	5	1,2
Клієнти	1*, 2*, 3	1,2,3	1
Доступ до стовпців (адреса, телефон, примітка)	5*	5	2
Працівники	1*	1	1
Замовлення	5*	5	1
Постачальники	1	1	1
Процедура «Періоцінка»	5	0	0
Процедура «Залишки»	5	0	5

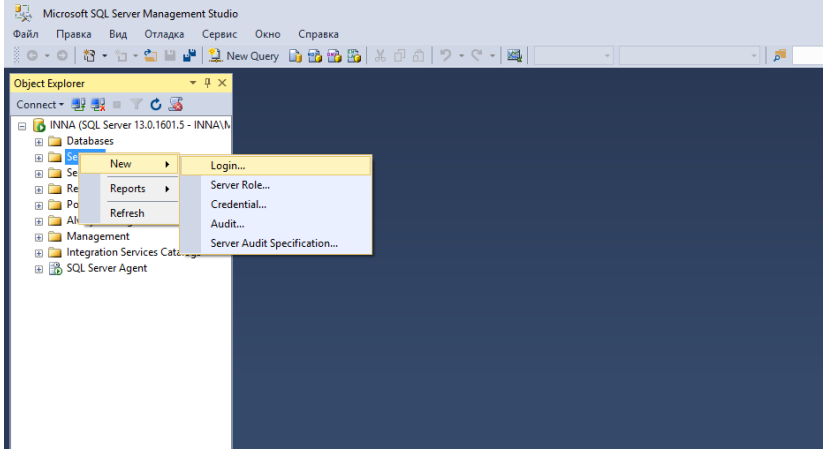
Створення користувачів

За допомогою інтерфейсних засобів:

Запускаєм програму «*Середовище SQLServerManagementStudio*»



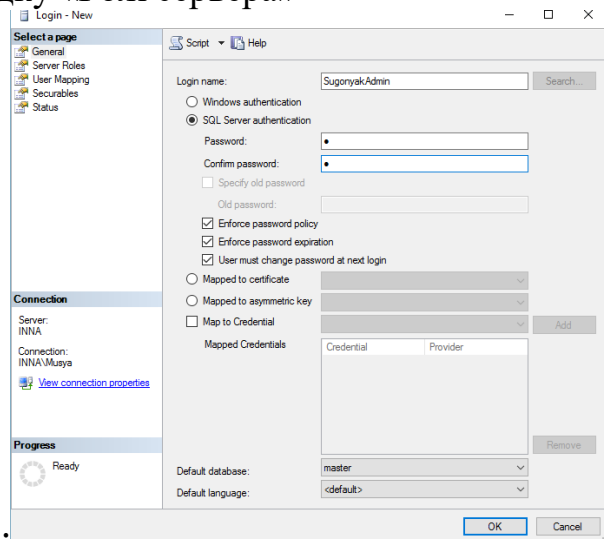
Якщо все введено вірно, у вікні «Оглядач об'єктів» ми побачимо вкладку з ім'ям нашого SQL-сервера. У ньому розкриваємо вкладку «Безпека» - «Імена входу» і в контекстному меню вибираємо «Створити ім'я входу».



Відкриється вікно «Створення імені входу». На вкладці «Загальні» заповнюємо:

- Ім’я входа: найменування користувача SQL.
- Перевірку достовірності вибираємо: SQL Server.
- Придумуємо пароль для користувача.

При необхідності можна визначити і інші налаштування безпеки). Потім переходимо на вкладку «Ролі сервера»



На даній сторінці необхідно вказати ролі сервера для даного користувача. Для створення користувача з адміністративними правами, необхідно встановити для нього роль

- sysadmin
  !!! Якщо створюється користувач для підключення програм, то достатньо вказівки ролей
- dbcreator
- processadmin

Роль public призначається всім користувачам.
Вказавши всі необхідні ролі для створюваного користувача натискаємо «ОК».На цьому процедура створення користувача завершена.

Створення користувача за допомогою TSQL

1. Для створення користувача з застосуванням T / SQL, потрібно скористатися трьома командами:

- інструкція Createlogin створює ім'я входу
- інструкція CreateUser створює користувача
- інструкція AlterRole і Grant дають права доступ

```
createlogintest_loginwithpassword='1'
usetest_base
createusertest_loginforlogintest_login

--роли БД
```

```
alterroledb_backupoperatoradd member test_login
execsp_helprolemember

--новая роль БД
createroletest_roleauthorizationdb_ddladmin
alterroledb_ddladminadd member test_login
alterroledb_ddladmindrop member test_login

--разрешения для роли
grantselecttotest_role

--разрешения для пользователя
grantselecttotest_login
revokeinserttotest_login
denydeletetotest_login
```

1. Для створення користувача з застосуванням T / SQL, потрібно скористатися трьома командами:

- інструкція Createlogin створює ім'я входу
- інструкція CreateUser створює користувача
- інструкція AlterRole і Grant дають права доступ

https://docs.microsoft.com/en-us/sql/t-sql/statements/grant-server-permissions-transact-sql или БД https://docs.microsoft.com/en-us/sql/t-sql/statements/grant-database-permissions-transact-sql).

Використання цих трьох інструкцій показано в наступному коді:

```
USE [master]
GO
CREATESERVERROLE [BulkAdmin] AUTHORIZATION [sa]
GO
ALTERSERVERROLE [BulkAdmin] ADD MEMBER [ISUGON]
GO
GRANT Administer Bulk Operations TO [SugonyakBulkAdmin]
GO
```

1. Для видалення користувача з ролі використовується інструкція AlterServerRole. При цьому використовується на параметр AddMember, a DropMember:

```
ALTER SERVER ROLEsysadminDROP MEMBERtestAdmin
```

!!!Аналогічним чином створюються ролі БД.

Перевірка прав доступу:

```
SELECTDB_NAME()AS'Database',
p.name,p.type_desc,p.is_fixed_role,dbp.state_desc,
```

```
dbp.permission_name, so.name,so.type_desc
FROMsys.database_permissionsdbp
LEFTJOINsys.objects so ONdbp.major_id=so.object_id
LEFTJOINsys.database_principals p ONdbp.grantee_principal_id=p.principal_id
--WHERE p.name = 'ProdDataEntry'
ORDERBY so.name,dbp.permission_name;
```

Перевірка застосування прав:

```
EXECUTEASLOGIN = 'ISUGON';
```

Потім ми повертаємося до дозволами sysadmin за допомогою наступної інструкції.

```
REVERT;
```

Перегляд ролей:
Ролі на сервері:
execsp_helpsrvrolemember
Ролі в БД
execsp_helpsrvrolemember

Створення користувача за допомогою системних процедур

Створення облікового запису
sp_addlogin
[@login=] 'обліковий_запис ‘
[, [@password=] 'пароль']
[, [@defdb=] 'база_даних_по_замовчуванню'];

Створення користувача
sp_adduser
[@loginame=] 'обліковий_запис ‘
[, [@name_in_db=] 'ім’я_користувача‘]
[, [@grpname=] 'ім’я_ролі'];

Створення нової ролі:
sp_addrole
[@rolename=] 'ім’я_ролі'
[, [@ownername=] 'ім’я__власника']
Додаваннякористувача до ролі:
spaddrolemember
[@rolename=] 'імя_ролі'
[@membername=] 'ім’я_користувача‘]

Присвоєння фіксованої ролі:
EXECsp_srvrolepermission 'sysadmin';

Операції присвоєння окремих прав виконуються або інструкцією Grantабо в діалоговому режимі.

Створення користувача

```
execsp_addlogin@loginame='test2_login', @passwd='2'

execsp_adduser'test2_login','test2_loginu'

grantselectONtariftotest2_loginu

--створення користувача


execsp_addroledf

grantcreatetabletodf
```

Перевірка

```
execsp_helpuser
```

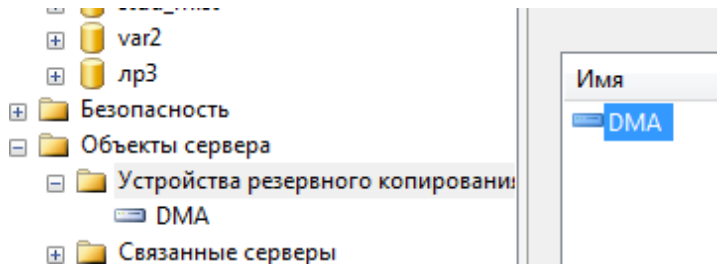
Завдання2. Резервування та відновлення БД

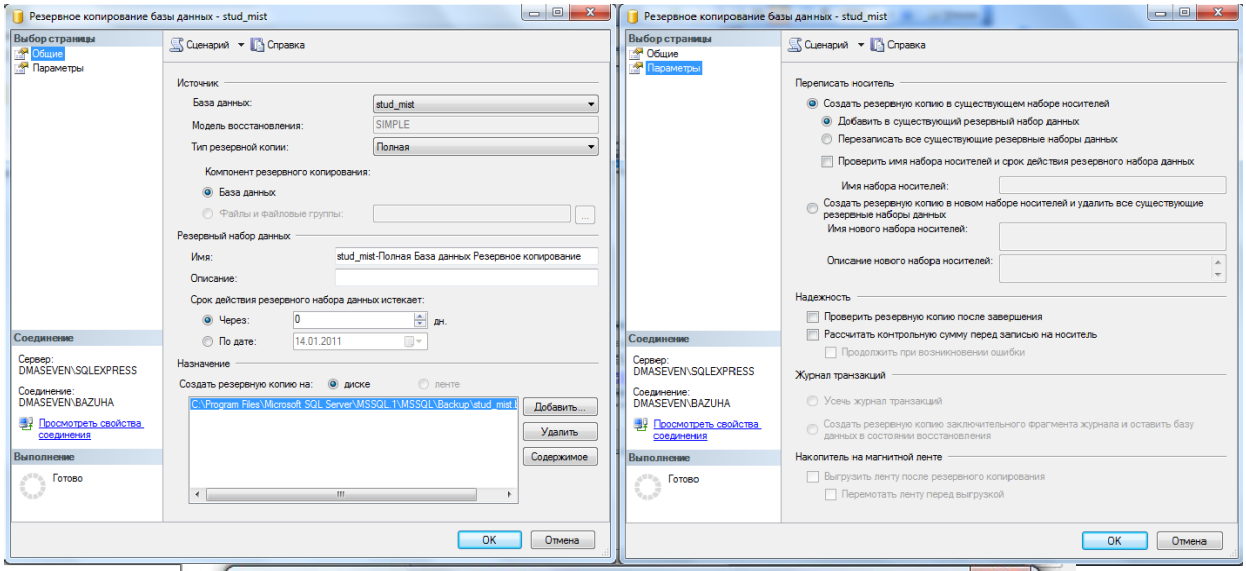
- створити резервні копії бази даних із використанням 3 стратегій резерування;
- виконати відновлення БД в усіх випадках.
- Звіт містить 3 файли – файл БД, лістинг запитів та процедур резервування із коментарями, різницевий одно файловий beckup.

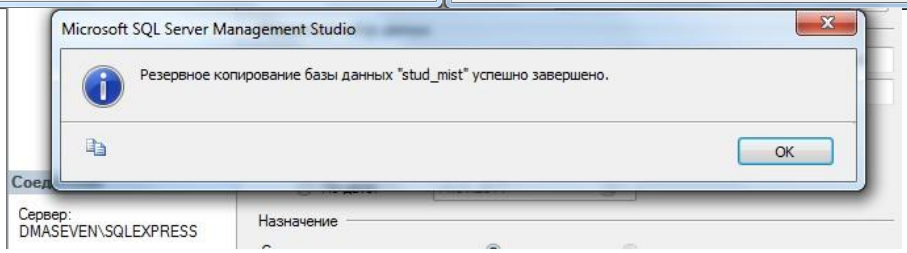
Приклади до опрацювання

Створення користувача за допомогою інтерфейсних засобів

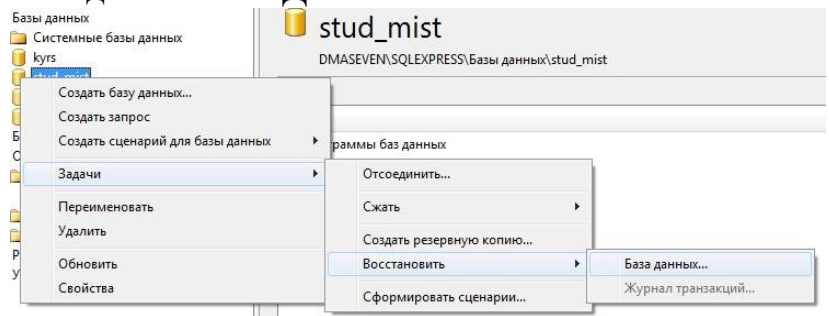
Проводимо резервування бази даних. Обираємо «Объекти сервера», «Устройство резервного копирования». Встановлюємо параметри резервування.

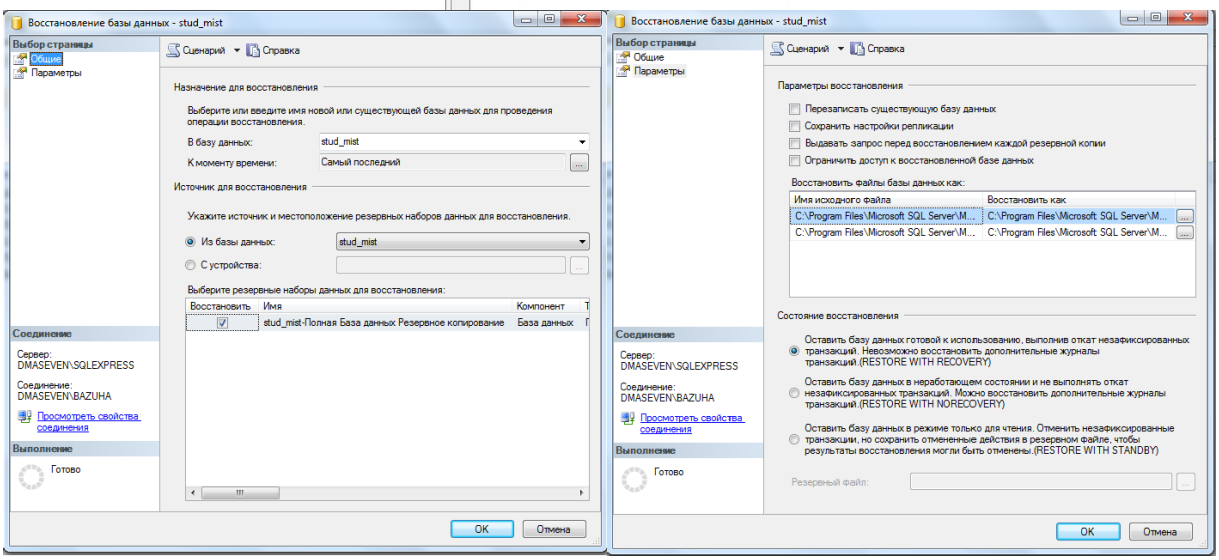


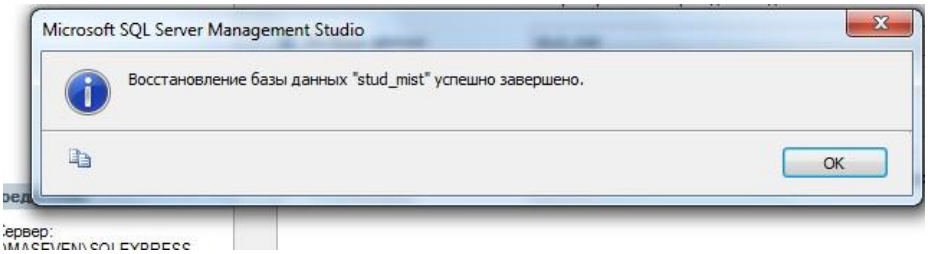




Проводимо відновлення БДStudMist







З використанням TSQL

Для виконання повного резервування бази даних виконайте наступний код:

```
BACKUP DATABASE AdventureWorks
TO DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_FullDbBkup.bak’ WITH
INIT, NAME = ‘AdventureWorks Full Db backup’,
DESCRIPTION = ‘AdventureWorksFullDatabaseBackup
```

Для повного відновлення бази даних виконайте наступну команду:

```
RESTORE DATABASE AdventureWorks
FROM DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_FullDbBkup.BAK’
WITHRECOVERY, REPLACE
```

Повний резервування не провадить очищення журналу транзакцій від неактивних записів. Якщо виконувати тільки повне резервування бази даних, слідом за цією операцією слід виконувати резервування файлу журналу з очищенням. Для цього використовується установка TRUNCATE_ONLY, як у наведеній нижче команді:

```
BACKUP LOG AdventureWorks
WITH TRUNCATE_ONLY
```

Для виконання повного резервування з журналом необхідно спершу виконати резервування всієї бази даних, як в наведеному нижче прикладі:

```
BACKUP DATABASE AdventureWorks
TO DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_FullDbBkup.bak’
WITH INIT, NAME = ‘AdventureWorks Full Db backup’,
DESCRIPTION = ‘AdventureWorksFullDatabaseBackup’
```

А потім слід виконати резервування журналу за допомогою команди:

```
BACKUP LOG AdventureWorks
TO DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_TlogBkup.bak’
WITH NOINIT, NAME = ‘AdventureWorksTranslog backup’,
DESCRIPTION = ‘AdventureWorks Transaction Log Backup’, NOFORMAT
```

Для відновлення з повної резервної копії або повної копії зі збереженням журналу виконайте наступні кроки.

1. Якщо база даних в змозі онлайн, обмежте доступ до неї, переключивши режим доступу (у вікні властивостей) на RESTRICTED_USER. Таким чином

доступ до бази даних буде дозволений тільки членам групи бази даних db_owner і членам груп сервера dbcreator і sysadmin.
2. Виконайте резервування заключного фрагмента журналу.
3. Виправте помилку, яка викликала крах бази даних.
4. Виконайте відновлення повної резервної копії з параметром NORECOVERY.
5. Якщо можливо, застосуєте всі збережені в резервних копіях журнали транзакцій з параметром NORECOVERY.
6. Виконайте відновлення резервної копії заключного фрагмента журналу з параметром RECOVERY.
Для виконання резервування заключного фрагмента журналу запустіть команду:

```
BACKUP LOG AdventureWorks
TO DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_TaillogBkup.bak’
WITHNORECOVER
```

Для повного відновлення повної резервної копії необхідно спочатку відновити файли бази даних за допомогою команди:

```
RESTORE DATABASE AdventureWorks
FROM DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_FullDbBkup.bak’
WITHNORECOVERY
```

Відразу за повним відновленням повинні бути відновлені всі резервні копії журналів транзакцій з параметром NORECOVERY, як показано нижче:

```
RESTORE LOG AdventureWorks
FROM DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_TlogBkup.bak’
WITHNORECOVERY
```

Нарешті, виконайте відновлення заключного фрагмента з параметром RECOVERY:

```
RESTORE LOG AdventureWorks
FROM DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_TaillogBkup.bak’
WITH RECOVERY
```

Збереження разностной резервної копії виконується командою:

```
BACKUP DATABASE AdventureWorks
TO DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_DiffDbBkup.bak’
WITH INIT, DIFFERENTIAL, NAME = ‘AdventureWorks Diff Db backup’,
DESCRIPTION = ‘AdventureWorks Differential Database Backup’
```

Щоб відновити базу даних з різницевої резервної копії, виконайте наступні кроки.
1. Якщо база даних в змозі онлайн, обмежте до неї доступ, переключивши режим доступу (у вікні властивостей) на RESTRICTED_USER. Тим самим доступ до бази даних буде дозволений тільки членам групи бази даних db_owner і членам груп сервера dbcreator і sysadmin.
2. Виконайте резервування заключного фрагмента журналу.

3. Виправте помилку, викликала збій бази даних.
4. Виконайте відновлення повної резервної копії з параметром NORECOVERY.
5. Виконайте відновлення останньої наявної разностной резервної копії з параметром NORECOVERY.
6. Виконайте відновлення резервної копії заключного фрагмента журналу з параметром RECOVERY.
Для відновлення разностной резервної копії (виконується після відновлення повної копії) введіть команду:

```
RESTORE DATABASE AdventureWorks
FROM DISK = ‘E:\SQLdata\BACKUPS\AdventureWorks_DiffDbBkup.bak’
WITH NORECOVERY
```

Завдання на самостійну роботу

Резервування баз даних є обмеженому колу осіб. За замовчуванням дозвіл дається членам певних груп системних адміністраторів серверів і ролям бази даних db_owner і db_backupoperator.

Лабораторна робота № 3

Створення плану обслуговування

Мета работи: дослідження задач і інструментарію серверів БД для організації регламентованого обслуговування.

Обладнаннятаінструментарій: ПК, MS SQL Server, MS Management Studio (Visual Studio)

Теоретичні відомості

Основи коротко: http://www.flenov.info/books.php?contentid=47

Помилково розглядати базу даних як якусь еталонну одиницю, оскільки, з плином часу, можуть проявлятися різного роду небажані ситуації - деградація продуктивності, збої в роботі та інше.

Для мінімізації ймовірності виникнення таких ситуацій створюють плани обслуговування із завдань, які гарантують стабільність і оптимальну продуктивність бази даних.

Серед подібних завдань можна виділити наступні:

1. Дефрагментація індексів
2. Оновлення статистики
3. Створення резервних копій

Крім фрагментації файлової системи і лог-файлу, відчутний вплив на продуктивність бази даних надає фрагментація всередині файлів даних:

1. Фрагментація всередині окремих сторінок індексу

Після операцій вставки, оновлення та видалення записів неминуче виникають порожні простору на сторінках. Нічого страшного в цьому немає, оскільки дана ситуація цілком нормальна, якби не одне але ...

Дуже важливу роль відіграє довжина рядка. Наприклад, якщо рядок має розмір, який займає більше половини сторінки, вільна половина цієї сторінки не буде використовуватися. В результаті при збільшенні числа рядків буде спостерігатися зростання невикористаного місця в базі даних.

Боротися з даним видом фрагментації стоїть на етапі проектування схеми, т. Е. Вибирати такі типи даних, які б компактно вміщалися на сторінках.

2. Фрагментація всередині структур індексу

Основна причина виникнення цього виду фрагментації - операції розбиття статті. Наприклад, відповідно до структури первинного ключа, новий рядок необхідно вставити на певну сторінку індексу, але цієї на сторінці недостатньо місця, щоб розмістити вставляються дані.

В такому випадку, створюється нова сторінка, на яку переміститися приблизно половина записів зі старої сторінки. Нова сторінка, часто не є фізично суміжній зі старою і, отже, позначається системою як фрагментована.

У будь-якому випадку, фрагментація веде до зростання числа сторінок для зберігання того ж обсягу інформації. Це автоматично призводить до збільшення розміру бази даних і зростання невикористаного місця.

При виконанні запитів, в яких йде звернення до фрагментованість індексам, потрібно більше IO операцій. Крім того, фрагментація накладає додаткові витрати на пам'ять самого сервера, якому доводиться зберігати в кеші зайві сторінки.

Для боротьби з фрагментацією індексів в арсеналі SQLServer передбачені команди: ALTERINDEXREBUILD / REORGANIZE.

Перебудова індексу має на увазі видалення старого і створення нового екземпляра індексу. Ця операція усуває фрагментацію, відновлює дисковий простір шляхом ущільнення сторінки, резервуючи при цьому вільне місце на сторінці, яке можна задати опцією FILLFACTOR. Важливо відзначити, що операція по перебудові індексу вельми затратна.

Тому, в разі, коли фрагментація незначна, переважно реорганізовувати існуючий індекс. Дана операція вимагає менших системних ресурсів, ніж пересозданіе індексу і полягає в реорганізації leaf-level сторінок. Крім того реорганізація при можливості стискає сторінки індексів.

Власне для чого потрібна статистика?

При виконанні будь-якого запиту, оптимізатор запитів, в рамках наявної у нього інформації, намагається побудувати оптимальний план виконання - який буде відображати з себе послідовність операцій, за рахунок виконання яких можна отримати необхідний результат, описаний в запиті.

В процесі вибору тієї чи іншої операції, оптимізатор запитів до числа найбільш важливих вхідних даних відносить статистику, що описує розподіл значень даних для стовпців усередині таблиці або індексу.

Така оцінка кількості елементів дозволяє оптимізатора запитів створювати більш ефективні плани виконання. У той же час, якщо статистикабуде містити застарілі дані, можуть бути обрані менш ефективні операції, які приведуть до створення повільних планів виконання. Наприклад, коли для невеликої вибірки на застарілої статистики вибирається більш витратний оператор IndexScan, замість оператора IndexSeek.

Як Ви бачите, щоб бути максимально корисною для оптимізатора запитів, статистика повинна бути точною і свіжою. Час від часу SQLServer періодично сам оновлює статистику - це поведінка регулюється опціями AUTO_CREATE_STATISTICS і AUTO_UPDATE_STATISTICS.

Крім того, при пересоздании індексів, статистика по ним оновлюється автоматично з включеним прапором FULLSCAN, що гарантує найбільш точний розподіл даних. При реорганізації індексів ж - статистика не оновлюється.

Коли дані в таблицях змінюються дуже часто, доцільно виконувати виборче оновлення статистики вручну, за допомогою операції UPDATESTATISTICS.

Також ручне оновлення, дуже важливо, коли для статистики заданий прапор NORECOMPUTE, що означає, що автоматичне оновлення статистики в подальшому не потрібно.

Завдання на лабораторну роботу

1. Створити скрипти основних процедур обслуговування БД
2. Налаштувати регламент виконання операцій і розклад для плану обслуговування, за допомогою SQL SERVER агент (або Планувальників завдань).
3. Налаштування розкладу резервування даних

Хід роботи

Завдання 1: Автоматична дефрагментація індексів

Ступінь фрагментації того чи іншого індексу можна дізнатися з динамічного системного уявлення sys.dm_db_index_physical_stats:

```
SELECT *
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, NULL)
WHERE avg_fragmentation_in_percent> 0
```

В даному запиті, останній параметр задає режим, від значення якого можливе швидке, але не зовсім точне визначення рівня фрагментації індексу (режими LIMITED / NULL). Тому рекомендується ставити режими SAMPLED / DETAILED.

Ми знаємо звідки отримати список фрагментованих індексів. Тепер необхідно для кожного з них згенерувати відповідну ALTERINDEX команду. Традиційно для цього використовують курсор:

```
DECLARE @SQL NVARCHAR(MAX)
DECLARE cur CURSOR LOCAL READ_ONLYFORWARD_ONLY FOR
        SELECT '
        ALTER INDEX [' + i.name + N'] ON [' +
SCHEMA_NAME(o.[schema_id]) + '].[' + o.name + '] ' +                CASE
WHEN s.avg_fragmentation_in_percent> 30
                    THEN 'REBUILD WITH (SORT_IN_TEMPDB = ON)'
                    ELSE 'REORGANIZE'
                END + ';'
        FROM (
                SELECT
                      s.[object_id]
                    , s.index_id
                    , avg_fragmentation_in_percent =
MAX(s.avg_fragmentation_in_percent)
                FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL,

NULL, NULL)     s

                WHERE s.page_count> 128 -- > 1 MB
```

```
                    AND s.index_id> 0 -- <> HEAP
                    AND s.avg_fragmentation_in_percent> 5
                GROUP BY s.[object_id], s.index_id
        ) s
        JOIN sys.indexesi WITH(NOLOCK) ON s.[object_id] = i.[object_id] AND
s.index_id = i.index_id
        JOIN sys.objects o WITH(NOLOCK) ON o.[object_id] = s.[object_id]
OPEN cur
FETCH NEXT FROM cur INTO @SQL
WHILE @@FETCH_STATUS = 0 BEGIN
        EXEC sys.sp_executesql  @SQL
        FETCH NEXT FROM cur INTO @SQL

END
CLOSEcur
DEALLOCATEcur
```

Щоб прискорити процес перевтілення індексу рекомендується додатково вказувати опцію SORT_IN_TEMPDB. Ще потрібно окремо згадати про опцію ONLINE - вона уповільнює пересозданіе індексу. Але іноді буває корисною. Наприклад, читання з кластерного індексу дуже дороге. Ми створили покриває індекс і вирішили проблему з продуктивністю. Далі ми робимо REBUILD некластерного індексу. У цей момент нам доведеться знову звертатися до кластерного індексу - що знижує перфоманс.
SORT_IN_TEMPDB дозволяє перебудовувати індекси в базі tempdb, що буває особливо корисно для великих індексів в разі нестачі пам'яті і іншому випадку - опція ігнорується. Крім того, якщо база tempdb розташована на іншому диску - це істотно скоротить час створення індексу. ONLINE дозволяє перебудувати індекс НЕ блокуючи при цьому запити до об'єкту для якого цей індекс створюється.
Як показала практика, дефрагментіроваліе індексів з низьким ступенем фрагментації або з невеликою кількістю сторінок не приносить яких-небудь помітних поліпшень, що сприяють підвищенню продуктивності при роботі з ними.
У доповненні, наведений вище запит можна переписати без застосування курсору:

```
DECLARE
     @IsDetailedScan BIT = 0
   , @IsOnline BIT = 0
DECLARE @SQL NVARCHAR(MAX)
SELECT @SQL = (
        SELECT '
        ALTER INDEX [' + i.name + N'] ON [' +
SCHEMA_NAME(o.[schema_id]) + '].[' + o.name + '] ' +
                CASE WHEN s.avg_fragmentation_in_percent> 30
                        THEN 'REBUILD WITH (SORT_IN_TEMPDB = ON'
```

```
                        -- Enterprise, Developer
                        + CASE WHEN SERVERPROPERTY('EditionID') IN
(1804890536, -2117995310) AND @IsOnline = 1
                                THEN ', ONLINE = ON'
                                ELSE ''
                          END +  ')'
                ELSE 'REORGANIZE'
        END + ';
    '
    FROM (
        SELECT
              s.[object_id]
            , s.index_id
            , avg_fragmentation_in_percent =
MAX(s.avg_fragmentation_in_percent)
        FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL,
NULL,
                                            CASE WHEN @IsDetailedScan
= 1
                                                THEN 'DETAILED'
                                                ELSE 'LIMITED'
                                            END) s
        WHERE s.page_count> 128 -- > 1 MB
            AND s.index_id> 0 -- <> HEAP
            AND s.avg_fragmentation_in_percent> 5
        GROUP BY s.[object_id], s.index_id
    ) s

    JOIN sys.indexesi ON s.[object_id] = i.[object_id] AND s.index_id =
i.index_id
    JOIN sys.objects o ON o.[object_id] = s.[object_id]
    FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
PRINT @SQL
EXECsys.sp_executesql @SQL
```

Щоб прискорити процес перевтілення індексу рекомендується В результаті обидва запити при виконанні будуть генерувати запити по дефрагментації проблемних індексів:

```
ALTER INDEX [IX_TransactionHistory_ProductID]
  ON [Production].[TransactionHistory] REORGANIZE;

ALTER INDEX
[IX_TransactionHistory_ReferenceOrderID_ReferenceOrderLineID]
  ON [Production].[TransactionHistory] REBUILD WITH (SORT_IN_TEMPDB =
ON, ONLINE = ON);

ALTER INDEX [IX_TransactionHistoryArchive_ProductID]
```

```
ON [Production].[TransactionHistoryArchive] REORGANIZE;
```
Власне, на цьому перша частина зі створення плану обслуговування для бази даних виконана.
Можливість дефрагментації окремих секцій:
```
USE ...
DECLARE
     @PageCount INT = 128
   , @RebuildPercent INT = 30
   , @ReorganizePercent INT = 10
   , @IsOnlineRebuild BIT = 0
   , @IsVersion2012Plus BIT =
       CASE WHEN CAST(SERVERPROPERTY('productversion') AS CHAR(2))
NOT IN ('8.', '9.', '10')
          THEN 1
          ELSE 0
       END
   , @IsEntEdition BIT =
       CASE WHEN SERVERPROPERTY('EditionID') IN (1804890536, -
2117995310)
          THEN 1
          ELSE 0
       END
   , @SQL NVARCHAR(MAX)
SELECT @SQL = (
   SELECT
'
ALTER INDEX ' + QUOTENAME(i.name) + ' ON ' + QUOTENAME(s2.name) +
'.' + QUOTENAME(o.name) + ' ' +
CASE WHEN s.avg_fragmentation_in_percent>= @RebuildPercent
          THEN 'REBUILD'
          ELSE 'REORGANIZE'
       END + ' PARTITION = ' +
       CASE WHEN ds.[type] != 'PS'
          THEN 'ALL'
          ELSE CAST(s.partition_number AS NVARCHAR(10))
END + ' WITH (' +
CASE WHEN s.avg_fragmentation_in_percent>= @RebuildPercent
          THEN 'SORT_IN_TEMPDB = ON' +
             CASE WHEN @IsEntEdition = 1
                   AND @IsOnlineRebuild = 1
                   AND ISNULL(lob.is_lob_legacy, 0) = 0
                   AND (
ISNULL(lob.is_lob, 0) = 0
                      OR
                         (lob.is_lob = 1 AND @IsVersion2012Plus = 1)
```

```
                )
              THEN ', ONLINE = ON'
              ELSE ''
            END
          ELSE 'LOB_COMPACTION = ON'
        END + ')'
    FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL,
NULL) s
    JOIN sys.indexesi ON i.[object_id] = s.[object_id] AND i.index_id = s.index_id
    LEFT JOIN (
        SELECT
            c.[object_id]
          , index_id = ISNULL(i.index_id, 1)
          , is_lob_legacy = MAX(CASE WHEN c.system_type_id IN (34, 35, 99)
THEN 1 END)
          , is_lob = MAX(CASE WHEN c.max_length = -1 THEN 1 END)
        FROM sys.columns c
        LEFT JOIN sys.index_columnsi ON c.[object_id] = i.[object_id]
          AND c.column_id = i.column_id AND i.index_id> 0
        WHERE c.system_type_id IN (34, 35, 99)
ORc.max_length = -1
GROUP BY c.[object_id], i.index_id
    ) lob ON lob.[object_id] = i.[object_id] AND lob.index_id = i.index_id
    JOIN sys.objects o ON o.[object_id] = i.[object_id]
    JOIN sys.schemass2 ON o.[schema_id] = s2.[schema_id]
    JOIN sys.data_spaces ds ON i.data_space_id = ds.data_space_id
    WHERE i.[type] IN (1, 2)
        AND i.is_disabled = 0
ANDi.is_hypothetical = 0
AND s.index_level = 0
        AND s.page_count> @PageCount
        AND s.alloc_unit_type_desc = 'IN_ROW_DATA'
        AND o.[type] IN ('U', 'V')
        AND s.avg_fragmentation_in_percent> @ReorganizePercent
    FOR XML PATH(''), TYPE
).value('.', 'NVARCHAR(MAX)')
PRINT @SQL
--EXEC sys.sp_executesql @SQL
```

Завдання 2. Оновлення статистики

Переглянути це властивість, як втім і на всі інші, можна у властивостях статистики:

```
SELECT s.*
```

```
FROM sys.stats s
JOIN sys.objects o ON s.[object_id] = o.[object_id]
WHERE o.is_ms_shipped = 0
```

Застосовуючи можливості динамічного SQL, напишемо скрипт з автоматичного оновлення застарілої статистики:

```
DECLARE @DateNowDATETIME
SELECT @DateNow = DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))
DECLARE @SQL NVARCHAR(MAX)
SELECT @SQL = (
   SELECT '
      UPDATE STATISTICS [' + SCHEMA_NAME(o.[schema_id]) + '].[' +
o.name + '] [' + s.name + ']
            WITH FULLSCAN' + CASE WHEN s.no_recompute = 1 THEN ',
NORECOMPUTE' ELSE '' END + ';'
      FROM sys.stats s WITH(NOLOCK)
      JOIN sys.objects o WITH(NOLOCK) ON s.[object_id] = o.[object_id]
      WHERE o.[type] IN ('U', 'V')
            AND o.is_ms_shipped = 0
            AND ISNULL(STATS_DATE(s.[object_id], s.stats_id),
GETDATE()) <= @DateNow
   FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
PRINT @SQL
EXECsys.sp_executesql @SQL
```

При выполнении будут генерироваться следующие стейтменты:

```
UPDATE STATISTICS [Production].[Shift] [PK_Shift_ShiftID] WITH
FULLSCAN;
UPDATE STATISTICS [Production].[Shift] [AK_Shift_Name] WITH
FULLSCAN, NORECOMPUTE;
```

Критерій старіння статистики в кожній конкретній ситуації може бути свій. В даному прикладі - 1 день.

У деяких випадках занадто часте оновлення статистики для великих таблиць може помітно знижувати продуктивність бази даних, а тому цей скрипт можна модифікувати. Наприклад, для великих таблиць оновлювати статистику рідше:

```
DECLARE @DateNowDATETIME
SELECT @DateNow = DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))
DECLARE @SQL NVARCHAR(MAX)
SELECT @SQL = (
   SELECT '
      UPDATE STATISTICS [' + SCHEMA_NAME(o.[schema_id]) + '].[' +
o.name + '] [' + s.name + ']
            WITH FULLSCAN' + CASE WHEN s.no_recompute = 1 THEN ',
NORECOMPUTE' ELSE '' END + ';'
      FROM (
```

```
		SELECT
			  [object_id]
			, name
			, stats_id
			, no_recompute
			, last_update = STATS_DATE([object_id], stats_id)
		FROM sys.stats WITH(NOLOCK)
		WHERE auto_created = 0
			AND is_temporary = 0 -- 2012+
	) s
	JOIN sys.objects o WITH(NOLOCK) ON s.[object_id] = o.[object_id]
	JOIN (
		SELECT
			  p.[object_id]
			, p.index_id
			, total_pages = SUM(a.total_pages)
		FROM sys.partitions p WITH(NOLOCK)
		JOIN sys.allocation_units a WITH(NOLOCK) ON p.[partition_id] =
a.container_id
		GROUP BY
			  p.[object_id]
			, p.index_id
	) p ON o.[object_id] = p.[object_id] AND p.index_id = s.stats_id
	WHERE o.[type] IN ('U', 'V')
		AND o.is_ms_shipped = 0
		AND (
			last_update IS NULL AND p.total_pages> 0 -- never updated
and contains rows
			OR
			last_update<= DATEADD(dd,
				CASE WHEN p.total_pages> 4096 -- > 4 MB
					THEN -2 -- updated 3 days ago
					ELSE 0
				END, @DateNow)
		)

   FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
PRINT @SQL
EXECsys.sp_executesql @SQL
```

Завданян 3. Автоматичне створення бекапів

Створимо план обслуговування резервного копіювання, а після обговоримо деякі тонкощі пов'язані з бекапу.

Створимо таблицю, в якій будуть записуватися повідомлення про помилки при створенні резервних копій:

```
USE [master]
GO
IF OBJECT_ID('dbo.BackupError', 'U') IS NOT NULL
   DROP TABLE dbo.BackupError
GO
CREATE TABLE dbo.BackupError (
dbSYSNAME PRIMARY KEY,
dtDATETIME NOT NULL DEFAULT GETDATE(),
msgNVARCHAR(2048)
)
GO
```

Скрипт для резервного копирования баз данных на каждый день я использую такой:

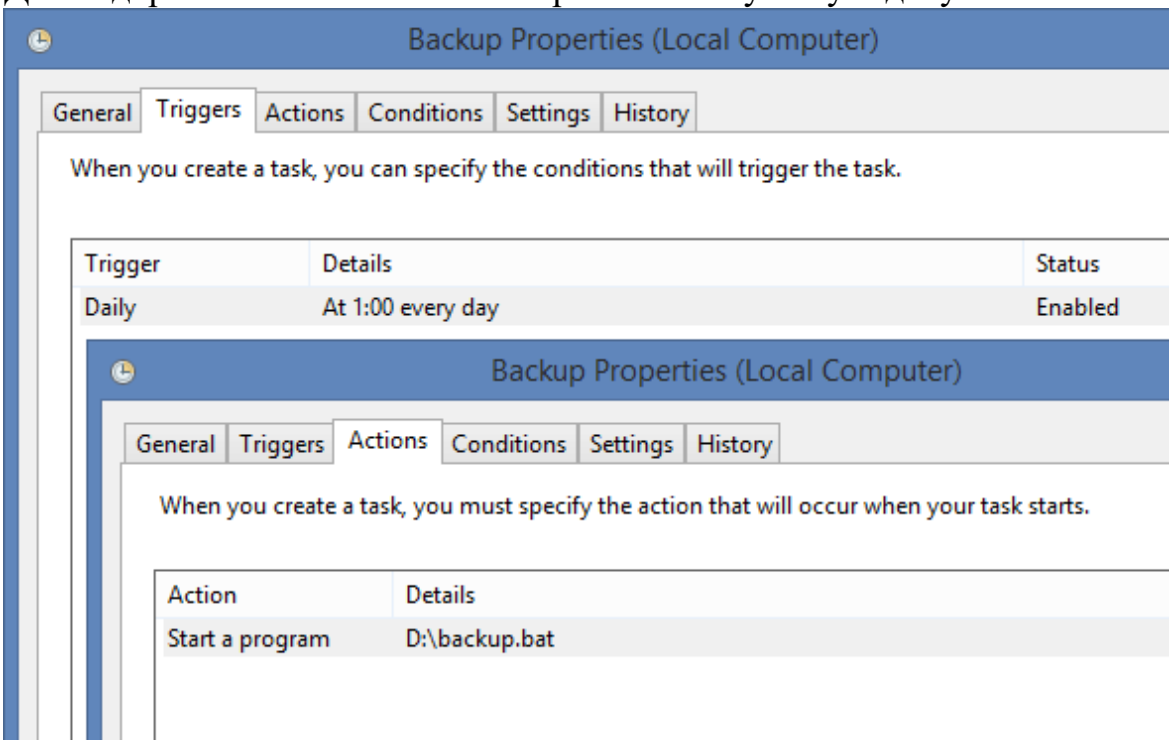
```
USE [master]
GO
SET NOCOUNT ON
TRUNCATE TABLE dbo.BackupError
DECLARE
     @dbSYSNAME
   , @sqlNVARCHAR(MAX)
   , @can_compress BIT
   , @path NVARCHAR(4000)
   , @name SYSNAME
   , @include_time BIT
--SET @path = '\\pub\backup' -- можно задать свой путь для бекапа
IF @pathISNULL -- либо писать в папку для бекапов указанную по умолчанию
EXEC [master].dbo.xp_instance_regread
N'HKEY_LOCAL_MACHINE',
N'Software\Microsoft\MSSQLServer\MSSQLServer',
N'BackupDirectory', @path OUTPUT, 'no_output'
SET @can_compress = ISNULL(CAST(( -- вопросысжатияобсуждаютсяниже
SELECT value
   FROM sys.configurations
   WHERE name = 'backup compression default') AS BIT), 0)
DECLARE cur CURSOR FAST_FORWARDREAD_ONLY LOCAL FOR
   SELECT d.name
   FROM sys.databases d
WHERE d.[state] = 0
AND d.name NOTIN ('tempdb') -- базы для которых не надо делать бекапов
OPEN cur
FETCH NEXT FROM cur INTO @db
WHILE @@FETCH_STATUS = 0 BEGIN
   IF DB_ID(@db) IS NULL BEGIN
```

```
      INSERT INTO dbo.BackupError (db, msg) VALUES (@db, 'db is missing')
   END
   ELSE IF DATABASEPROPERTYEX(@db, 'Status') != 'ONLINE' BEGIN
      INSERT INTO dbo.BackupError (db, msg) VALUES (@db, 'db state !=
ONLINE')
   END
   ELSE BEGIN
      BEGIN TRY
         SET @name = @path + '\T' + CONVERT(CHAR(8), GETDATE(), 112) +
'_' + @db + '.bak'
         SET @sql = '
            BACKUP DATABASE ' + QUOTENAME(@db) + '
            TO DISK = ''' + @name + ''' WITH NOFORMAT, INIT' +
            CASE WHEN @can_compress = 1 THEN ', COMPRESSION' ELSE ''
END
         --PRINT @sql
         EXEC sys.sp_executesql @sql
      END TRY
      BEGIN CATCH
         INSERT INTO dbo.BackupError (db, msg) VALUES (@db,
ERROR_MESSAGE())
      END CATCH
   END
   FETCH NEXT FROM cur INTO @db
END
CLOSEcur
DEALLOCATEcur
```

Якщо на сервері налаштований компонент DatabaseMail, то в скрипт можна додати повідомлення поштою про виниклі проблеми:

```
IF EXISTS(SELECT 1 FROM dbo.BackupError) BEGIN
   DECLARE @report NVARCHAR(MAX)
   SET @report =
      '<table
border="1"><tr><th>database</th><th>date</th><th>message</th></tr>' +
CAST((
         SELECT td = db, '', td = dt, '', td = msg
         FROM dbo.BackupError
         FOR XML PATH('tr'), TYPE
      ) AS NVARCHAR(MAX)) +
      '</table>'
   EXEC msdb.dbo.sp_send_dbmail
      @recipients = 'your_account@mail.ru',
      @subject = 'Backup Problems',
      @body = @report,
@body_format = 'HTML'
```

END
Власне, на цьому етапі, робочий скрипт для автоматичного створення резервних копій готовий. Залишається його створити job, який би за розкладом запускав цей скрипт.
Власників Express редакцій потрібно окремо згадати, оскільки в SQLServerExpressedition немає можливості використовувати SQLServerAgent. Яка б печалька не прийшла після цих слів, насправді, все можна вирішити. Найпростіше створити bat файл з приблизно схожим змістом:
sqlcmd -S <ComputerName> \ <InstanceName> -iD: \ backup.sql
Далі відкрити TaskScheduler і створити в ньому нову задачу.



Друга альтернатива - використовувати сторонні розробки, які дозволяють запускати завдання за розкладом. Серед можна виділити SQLScheduler - зручний і безкоштовний тул. Інсталятор у мене загубився, тому буду вдячний, якщо хтось поділитися робочої посиланням для читачів.
Тепер поговоримо про корисні дрібниці пов'язаних з бекапу.
Стиснення ...
Можливість стиснення бекапів з'явилася вперше в SQLServer 2008. Згадую з ностальгією час, коли працюючи на 2005 версії мені доводилося 7Zip-му стискати бекапи. Тепер же все стало набагато простіше.
Але потрібно пам'ятати, що стиснення бекапів буде використовуватися тільки якщо виконувати команду BACKUPс параметром COMPRESSION або включити стиск за замовчуванням за допомогою такої команди:

```
USE [master]
GO
EXEC sp_configure 'backup compression default', 1
RECONFIGUREWITHOVERRIDE
GO
```

До слова буде сказано, що стислі бекапи має деякі переваги: потрібно менше місця для їх зберігання, відновлення БД з стислих бекапів зазвичай виконується трішки швидше, також вони швидше створюються, оскільки вимагають меншої кількості I / O операцій. Мінуси, до речі, теж є - при роботі зі стиснутими бекап навантаження на процесор збільшується.
Цим запитом можна повернути розмір останнього FULL бекапу із стисненням і без:

```
SELECT
database_name
   , backup_size_mb = backup_size / 1048576.0
   , compressed_backup_size_mb = compressed_backup_size / 1048576.0
   , compress_ratio_percent = 100 - compressed_backup_size * 100. / backup_size
FROM (
  SELECT
database_name
       , backup_size
       , compressed_backup_size = NULLIF(compressed_backup_size,
backup_size)
       , RowNumber = ROW_NUMBER() OVER (PARTITION BY database_name
ORDER BY backup_finish_dateDESC)
   FROM msdb.dbo.backupset
   WHERE [type] = 'D'
) t
WHEREt.RowNumber = 1
```

Зазвичай стиснення досягає 40-90%, а то й брати до уваги бінарні дані.
Якщо модифікувати попередній запит, то можна моніторити для яких баз робилися резервні копії:

```
SELECT
        d.name
   , rec_model = d.recovery_model_desc
   , f.full_time
   , f.full_last_date
   , f.full_size
   , f.log_time
   , f.log_last_date
   , f.log_size
FROM sys.databases d
LEFT JOIN (
   SELECT
       database_name
         , full_time = MAX(CASE WHEN [type] = 'D' THEN
CONVERT(CHAR(10), backup_finish_date - backup_start_date, 108) END)
```

```
        , full_last_date = MAX(CASE WHEN [type] = 'D' THEN
backup_finish_date END)
        , full_size = MAX(CASE WHEN [type] = 'D' THEN backup_size END)
        , log_time = MAX(CASE WHEN [type] = 'L' THEN
CONVERT(CHAR(10), backup_finish_date - backup_start_date, 108) END)
        , log_last_date = MAX(CASE WHEN [type] = 'L' THEN
backup_finish_date END)
        , log_size = MAX(CASE WHEN [type] = 'L' THEN backup_size END)
    FROM (
        SELECT
            s.database_name
            , s.[type]
            , s.backup_start_date
            , s.backup_finish_date
            , backup_size =
                    CASE WHEN s.backup_size =
s.compressed_backup_size
                            THEN s.backup_size
                            ELSE s.compressed_backup_size
                    END / 1048576.0
            , RowNum = ROW_NUMBER() OVER (PARTITION BY
s.database_name, s.[type] ORDER BY s.backup_finish_dateDESC)
        FROM msdb.dbo.backupset s
        WHERE s.[type] IN ('D', 'L')
    ) f
    WHERE f.RowNum = 1
    GROUP BY f.database_name
) f ON f.database_name = d.name
```

Якщо у Вас SQLServer 2005, то цей рядок:

```
backup_size = CASE WHEN s.backup_size = s.compressed_backup_size THEN
...
```

потрібно поміняти на: `backup_size = s.backup_size / 1048576.0`

Результати цього запиту можуть допомогти запобігти багато проблем:

Можна відразу побачити, що для всіх чи БД є FULL бекапи за актуальну дату.

Далі можна подивитися на час створення бекапа. Навіщо питається? Припустимо, що раніше бекап бази DB_Dev займав 5 секунд, а потім став займати 1 годину. Причин цього може бути багато: диски не справляються з навантаженням, дані в базі виросли до непристойних розмірів, полетів диск в RAID і швидкість запису знизилася.

Якщо у бази варто модель відновлення FULL або BULK_LOGGED, то бажано час від часу робити бекап балки, щоб не прирікати сервер на муки постійного зростання LDF файлу. Ступінь заповнення файлу даних і балки для баз даних можна подивитися цим запитом:

```
IF OBJECT_ID('tempdb.dbo.#space') IS NOT NULL
   DROP TABLE #space
CREATE TABLE #space (
database_id INT PRIMARY KEY,
data_used_sizeDECIMAL(18,6),
log_used_sizeDECIMAL(18,6)
)
DECLARE @SQL NVARCHAR(MAX)
SELECT @SQL = STUFF((
   SELECT '
   USE [' + d.name + ']
   INSERT INTO #space (database_id, data_used_size, log_used_size)
   SELECT
        DB_ID()
         , SUM(CASE WHEN [type] = 0 THEN space_used END)
         , SUM(CASE WHEN [type] = 1 THEN space_used END)
   FROM (
         SELECT s.[type], space_used = SUM(FILEPROPERTY(s.name,
"SpaceUsed") * 8. / 1024)
         FROM sys.database_files s
         GROUP BY s.[type]
   ) t;'
   FROM sys.databases d
   WHERE d.[state] = 0
   FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1, 2, '')
EXEC sys.sp_executesql @SQL
SELECT
        database_name = DB_NAME(t.database_id)
   , t.data_size
   , s.data_used_size
   , t.log_size
   , s.log_used_size
   , t.total_size
FROM (
   SELECT
        database_id
         , log_size = SUM(CASE WHEN [type] = 1 THEN size END) * 8. / 1024
         , data_size = SUM(CASE WHEN [type] = 0 THEN size END) * 8. / 1024
         , total_size = SUM(size) * 8. / 1024
   FROM sys.master_files
   GROUP BY database_id
) t
LEFT JOIN #space s ON t.database_id = s.database_id
```

Результати запиту на моєму локальному інстанси:

Ще хотів показати пару цікавих трюків, які можуть полегшити життя. Якщо при виконанні команди BACKUP вказати кілька шляхів, то кінцевий файл з бекапу буде розрізано на шматки приблизно однакового розміру.

```
BACKUP DATABASE AdventureWorks2012
TO
        DISK = 'D:\AdventureWorks2012_1.bak',
        DISK = 'D:\AdventureWorks2012_2.bak',
        DISK = 'D:\AdventureWorks2012_3.bak'
```

Одного разу мені це стало в нагоді, коли довелося копіювати бекап на флешку з файлової системою FAT32, в якій є обмеження на максимальний розмір файлу.

Ще одна цікава можливість - створювати копію бекапу. З особистого досвіду скажу, що доводилося зустрічати людей, які спочатку створювали бекап в дефолтной папці, а потім руками або скриптом копіювали на дискову кулі. А потрібно було просто використовувати таку команду:

```
BACKUPDATABASEAdventureWorks2012
        TODISK = 'D:\AdventureWorks2012.bak'
        MIRROR TO DISK = 'E:\AdventureWorks2012_copy.bak'
        WITHFORMAT
```

Завдання на самостійну роботу

Розглянути стратегію обслуговування баз даних за допомогою скриптових засобів PowerShell або Bush

Лабораторна робота № 4

Дослідження роботи SSIS

Мета: освоїти інструментарій інтеграції даних корпоративних інформаційних систем

ПО і інструменти: необхідно встановити SSDT (SSQTBI), поддежівающіе наступні версії SQL Server: SQL Server 2014 року, SQL Server 2012 SQL Server 2008 і 2008 R2.

Теоретичні відомості

SSIS - це інструмент, який дозволяє в зручному вигляді реалізувати інтеграцію, тобто реалізувати процес перенесення даних з одного джерела в інший. Цей процес іноді називають ETL (від англ. Extract, Transform, Load - дослівно «витяг, перетворення, завантаження»).

Необхідні інструменти для вивчення SSIS

SSIS буде розглядатися на прикладі SQL Server 2014 Developer Edition. Служби Integration Services доступні SQL Server 2014 начінаясредакціі Standard.

Додатково необхідно буде завантажити і встановити інструмент розробника SQL Server Data Tools (SSDT).

SSDT - це розширення для Visual Studio, яке дозволить створювати проекти необхідного нам типу.

Для полегшення процесу установки, я скористаюся SSDT для Visual Studio 2012 (VS2012), його можна завантажити за посиланням (файл «SSDTBI_VS2012_x86_ENU.exe»): www.microsoft.com/en-US/download/details.aspx?id=36843

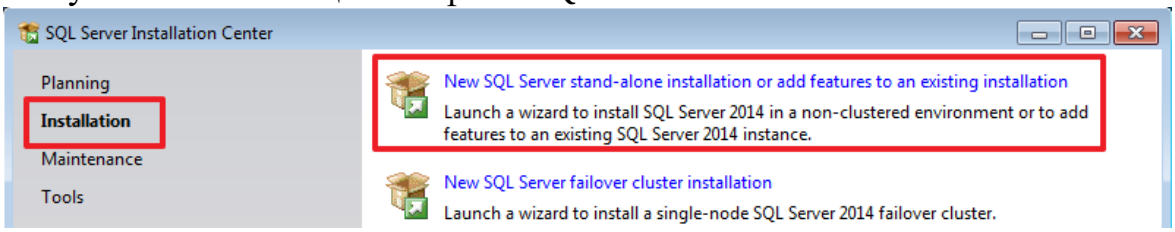
Якщо на вашому комп'ютері не встановлено VS даної версії, то установник SSDT встановить мінімальну версію оболонки, яка дозволить створювати проекти потрібного нам типу.

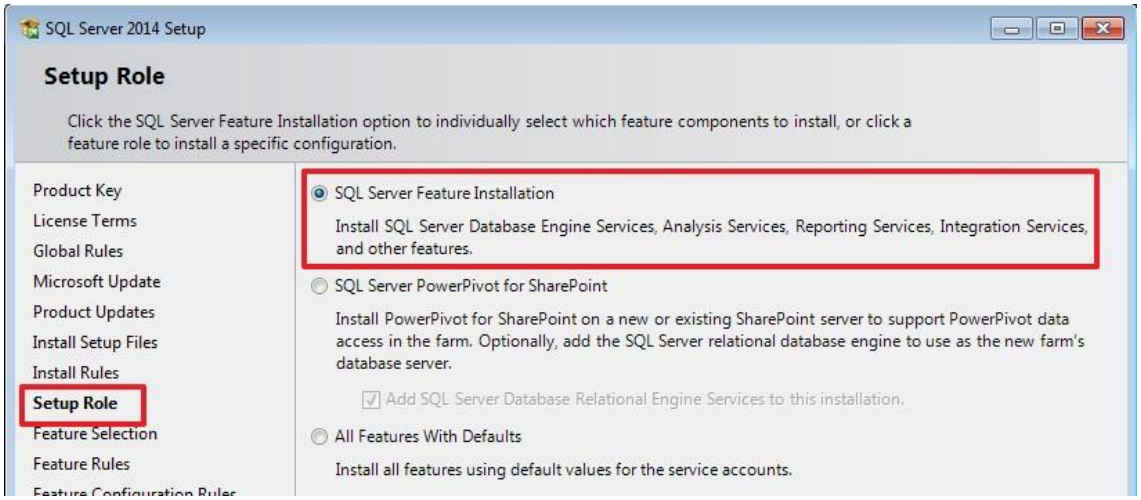
Встановлення SQL Server і SSDT

Насамперед встановимо SQL Server з усіма необхідними компонентами. При наявність ОС Windows 7 SP 1 (x64), нічого додаткового крім зазначеного нижче встановлювати не доведеться.

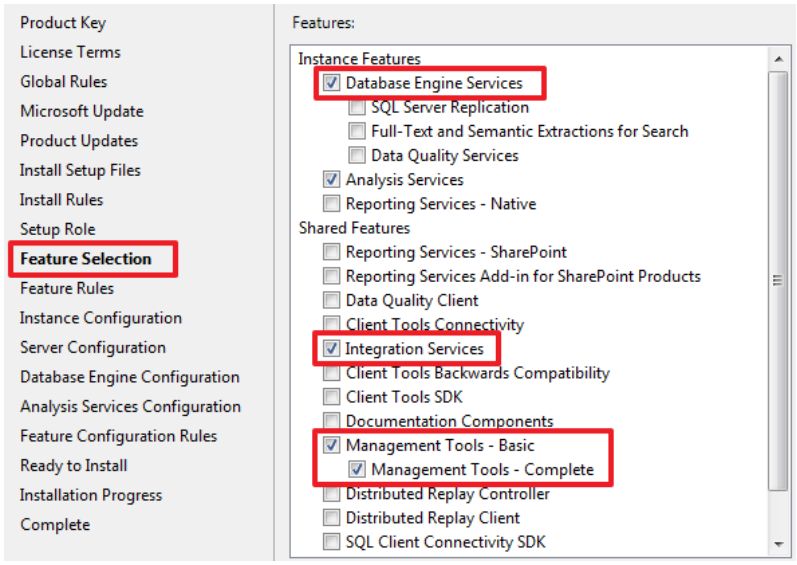
Оскільки курс призначений для початківців, то розпишу весь процес установки детально.

Запускаємо інсталяційний файл SQL Server 2014:



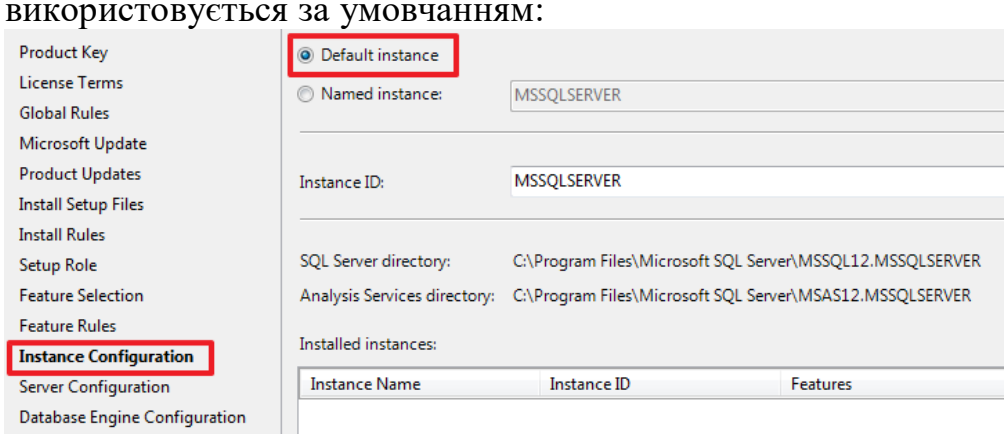


Для роботи SSIS досить буде вибрати такі компоненти:

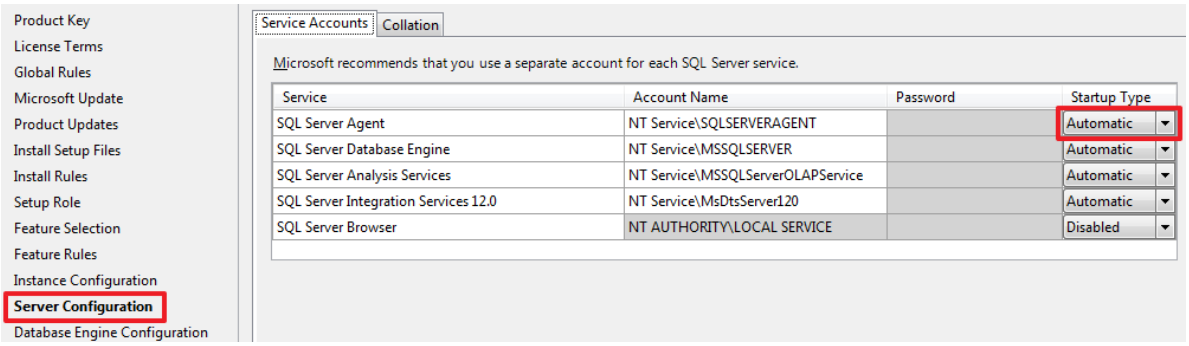


Оскільки мені в подальшому знадобиться Analysis Services (SSAS), то я відзначив і його, якщо він вам не потрібен ви можете не вибирати даний компонент.

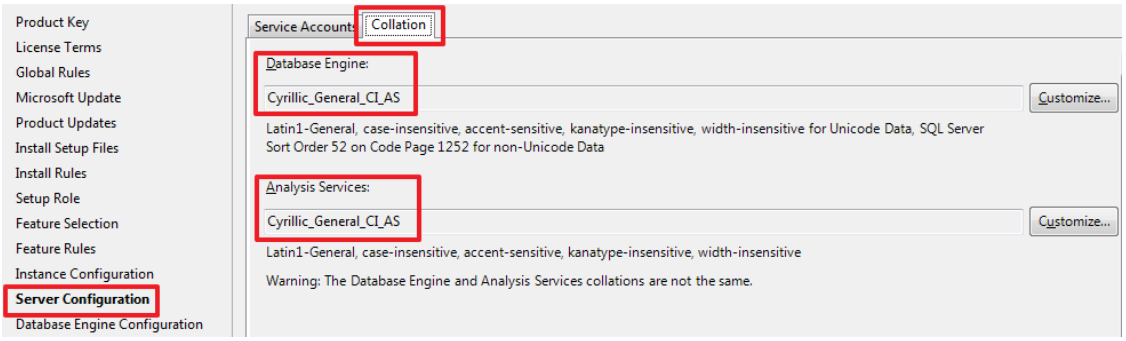
У мене немає інших встановлених SQL Server, і я зроблю цей екземпляр використовується за умовчанням:



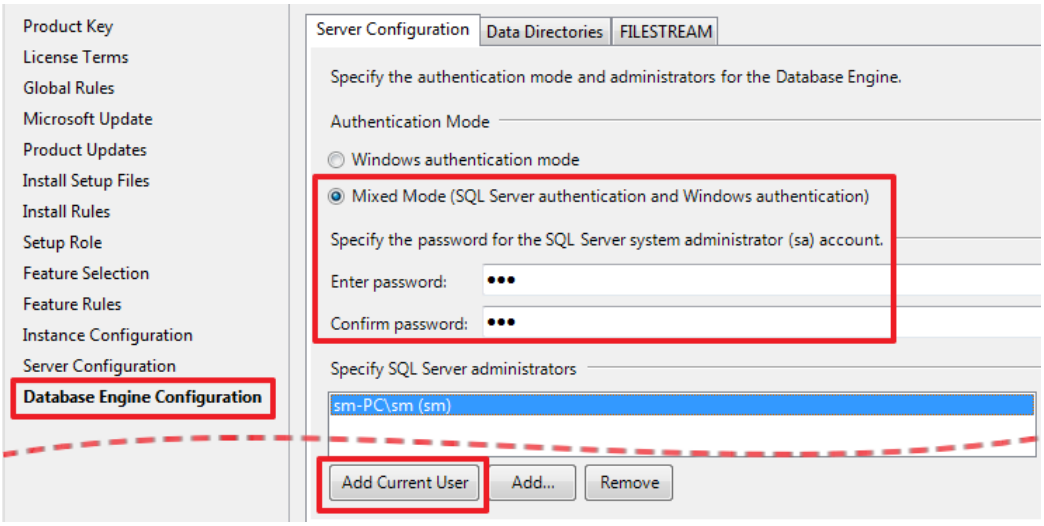
Зроблю, щоб SQL Agent запускався автоматично:



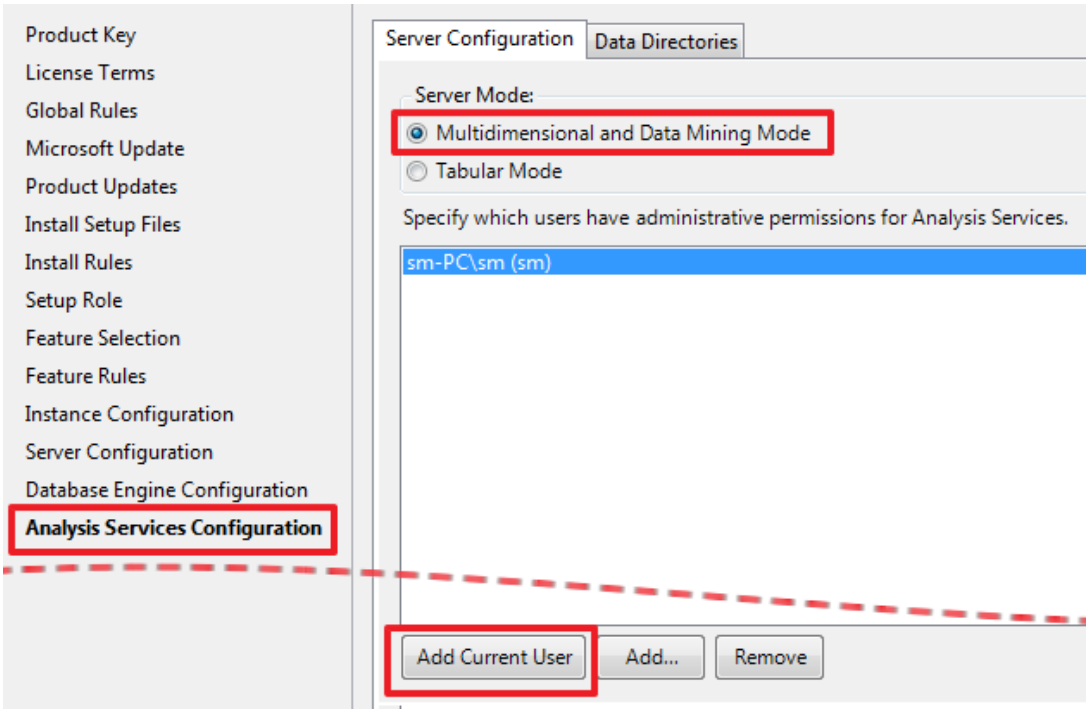
При необхідності можна змінити Collation, який буде використовуватися за замовчуванням:



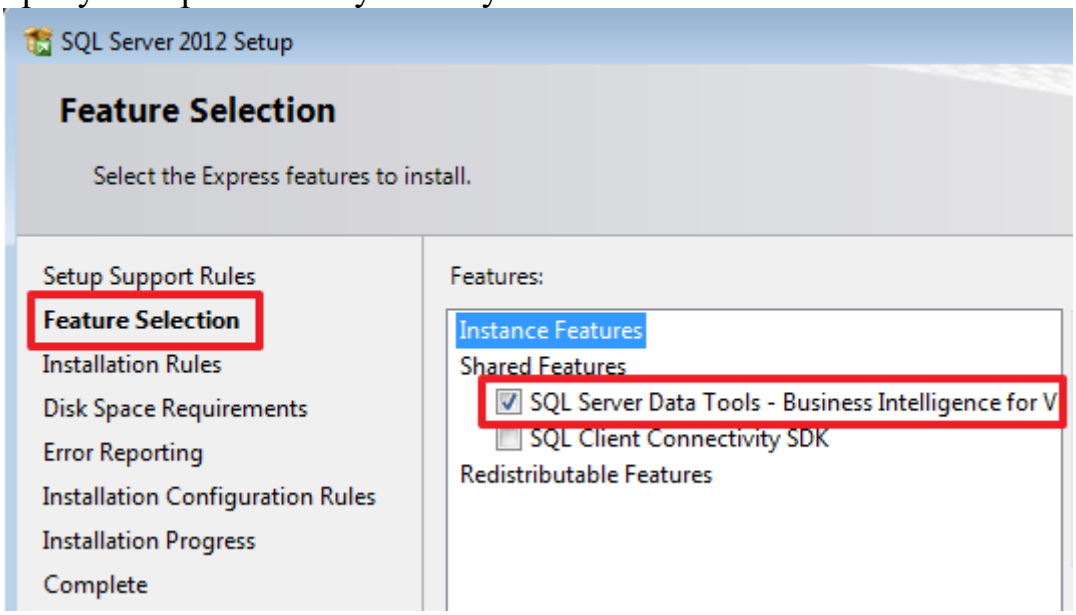
Встановлю змішаний режим аутентифікації, вказавши свій пароль для користувача sa:



Оскільки я ще вибрав Analysis Services, то роблю настройки для нього:



Натискаючи Next і Install запускаємо установку SQL Server і його компонент.Так як у мене на комп'ютері всього один диск, то все директорії я залишив за замовчуванням, при необхідності ви можете змінити їх на більш зручні.
Наступним кроком встановимо SSDT - це розширення для Visual Studio, яке дасть нам можливість створювати проекти SSIS.
Установник SSDT ставить мінімальну версію оболонки VS, тому попередньо встановлювати VS окремо немає потреби.
Запускаємо «SSDTBI_VS2012_x86_ENU.exe», і діставшись до наступного кроку вибираємо наступний пункт:



Натискаючи Next запускаємо установку.

Після завершення установки на всякий випадок перезавантажуємо комп'ютер.

Завдання на лабораторну роботу

1. Реалізувати інтеграцію 2 джерел даних і імпорт даних в БД.
2. Індивідуальне завдання. Реалізувати заповнення БД з альтернативного джерела (таблиці Excelілі MySQLServer)

Хід роботи

Створення демонстраційних баз даних
Запустимо SQL Server Management Studio (SSMS) і за допомогою скрипта створимо 3 бази даних - перші дві (DemoSSIS_SourceA і DemoSSIS_SourceB) виступатимуть в ролі джерел даних, а третя (DemoSSIS_Target) в ролі одержувача даних:

```
-- перша БД виступає в ролі джерела даних
CREATEDATABASE DemoSSIS_SourceA
GO

ALTERDATABASE DemoSSIS_SourceA SETRECOVERY SIMPLE
GO

-- друга БД виступає в ролі джерела даних
CREATEDATABASE  DemoSSIS_SourceB
GO

ALTERDATABASE DemoSSIS_SourceB SETRECOVERY SIMPLE
GO

-- БД виступає в ролі одержувача даних
CREATEDATABASE DemoSSIS_Target
GO

ALTERDATABASE DemoSSIS_Target SETRECOVERY SIMPLE
GO
```

У базах джерелах створимо тестові таблиці і наповнимо їх тестовими даними:

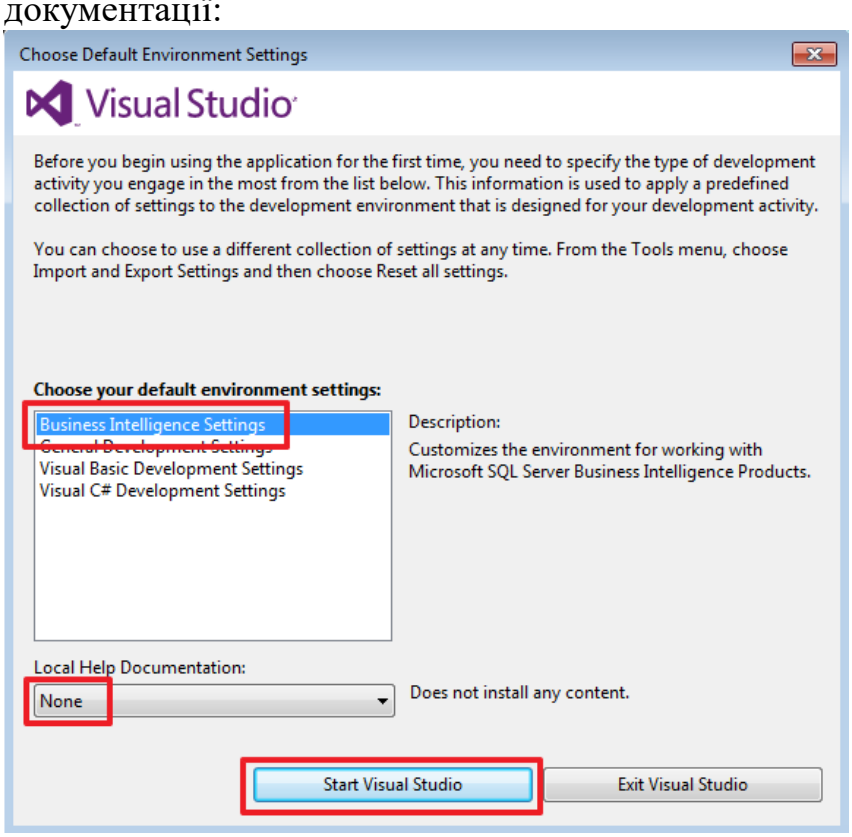
```
USE DemoSSIS_SourceA
GO

-- продукти з джерела A
CREATETABLEProducts(
```

```
IDintNOTNULLIDENTITY,
  Title nvarchar(50) NOTNULL,
  Price money,
CONSTRAINT PK_Products PRIMARY KEY(ID)
)
GO

-- наповнюємо таблицю тестовими даними
SET IDENTITY_INSERT Products ON

INSERT Products(ID,Title,Price)VALUES
(1,N'Клей',20),
(2,N'Корректор',NULL),
(3,N'Скотч',100),
(4,N'Стикеры',80),
(5,N'Скрепки',25)

SET IDENTITY_INSERT Products OFF
GO


USE DemoSSIS_SourceB
GO

-- продукти з джерела B
CREATETABLEProducts(
IDintNOTNULLIDENTITY,
  Title nvarchar(50) NOTNULL,
  Price money,
CONSTRAINT PK_Products PRIMARY KEY(ID)
)
GO

-- наповнюємо таблицю тестовими даними
SET IDENTITY_INSERT Products ON

INSERT Products(ID,Title,Price)VALUES
(1,N'Ножницы',200),
(2,N'Ножканцелярский',70),
(3,N'Дырокол',220),
(4,N'Степлер',150),
(5,N'Шариковаяручка',15)

SET IDENTITY_INSERT Products OFF
GO
```

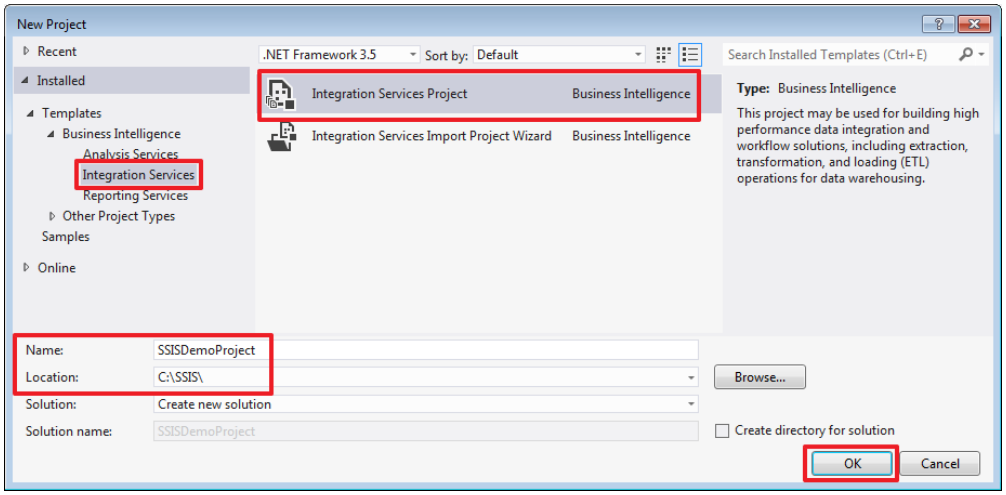
Створимо таблицю в приймаючій базі:

```
USE DemoSSIS_Target
GO

-- приймаюча таблиця
CREATETABLEProducts(
IDintNOTNULLIDENTITY,
  Title nvarchar(50) NOTNULL,
  Price money,
SourceID char(1) NOTNULL, -- використовується для ідентифікації джерела
SourceProductID intNOTNULL, -- ID в джерелі
CONSTRAINT PK_Products PRIMARY KEY(ID),
CONSTRAINT UK_Products UNIQUE(SourceID,SourceProductID),
CONSTRAINT CK_Products_SourceID CHECK(SourceID IN('A','B'))
)
GO
```

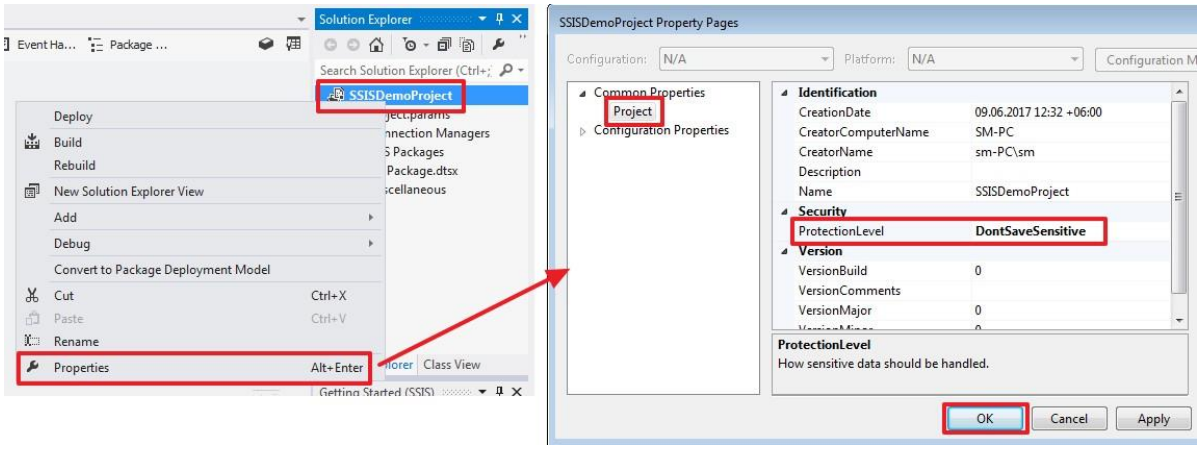
Створення SSIS проекту
Запустимо Visual Studio 2012 і виберемо один з видів пропонованих нам налаштувань середовища, в цьому ж місці я відмовлюся від локальної документації:



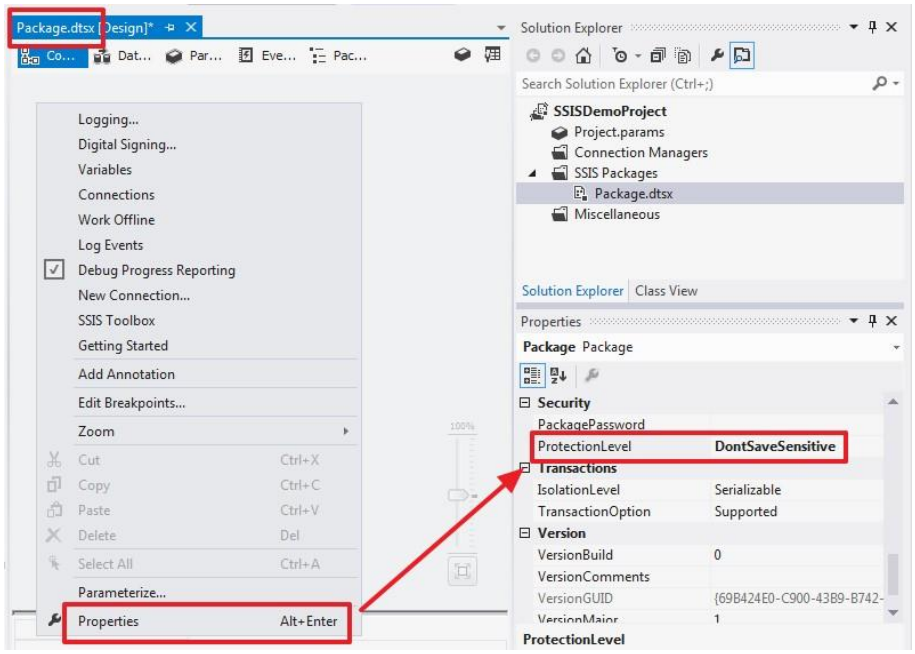
Створимо новий проект (**File -> New -> Project…**):



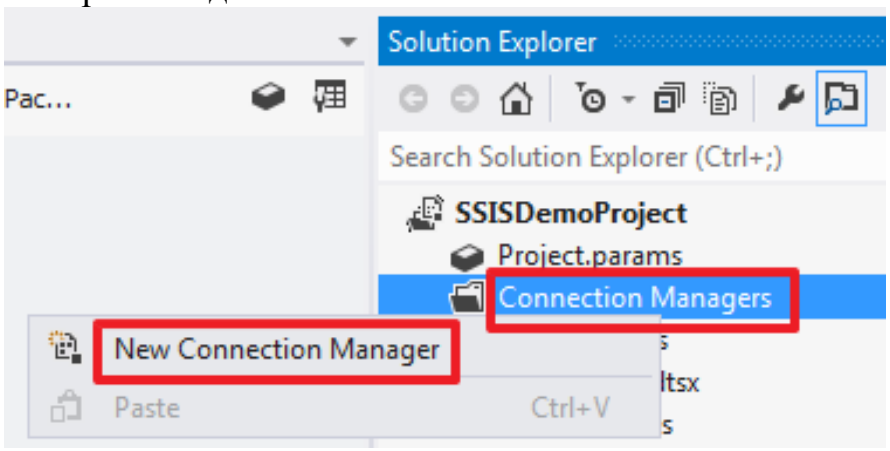
Для подальшого полегшення розгортання зайдемо в властивості проекту та змінимо опцію **ProtectionLevel** на **DontSaveSensitive**:

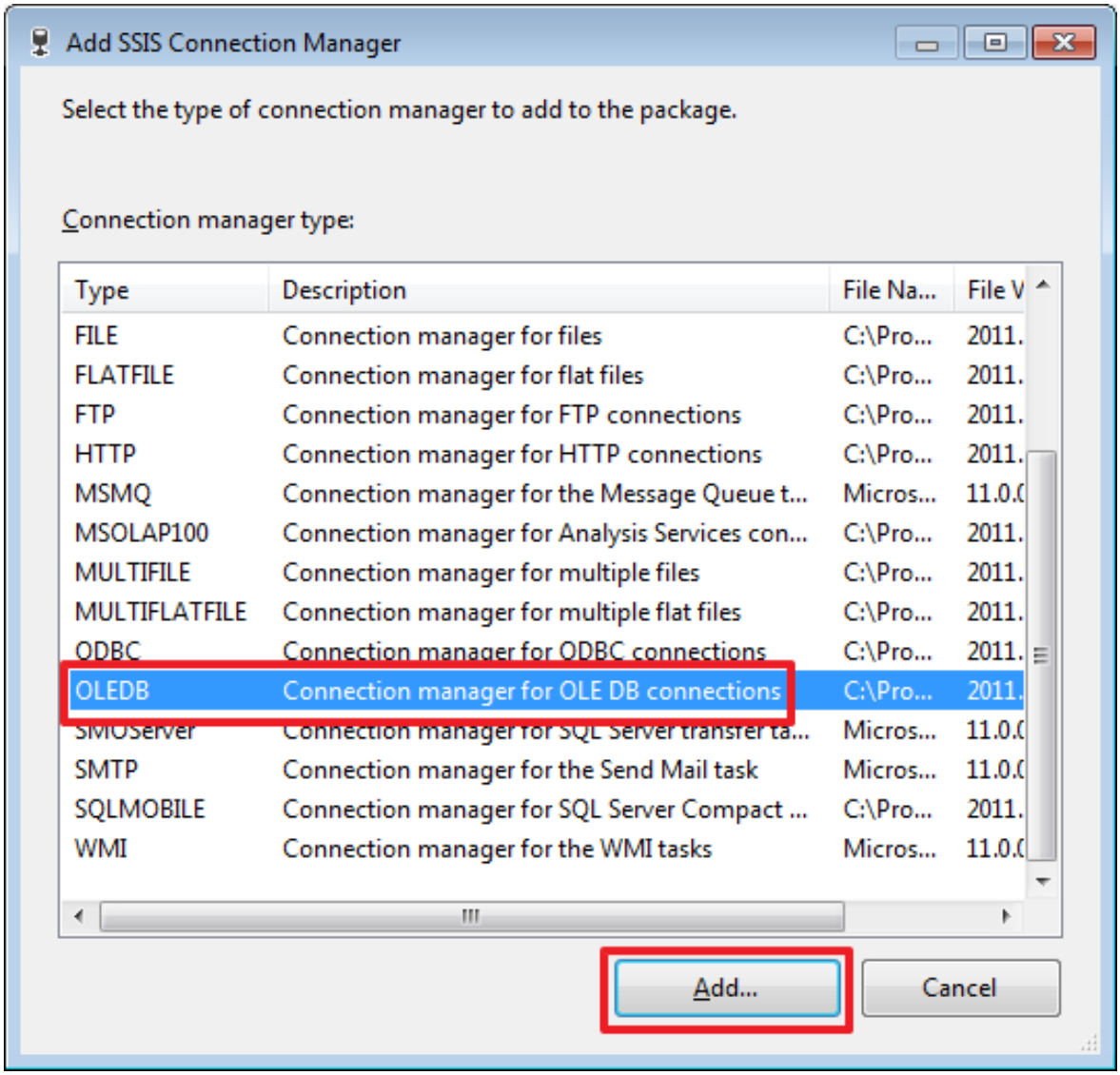


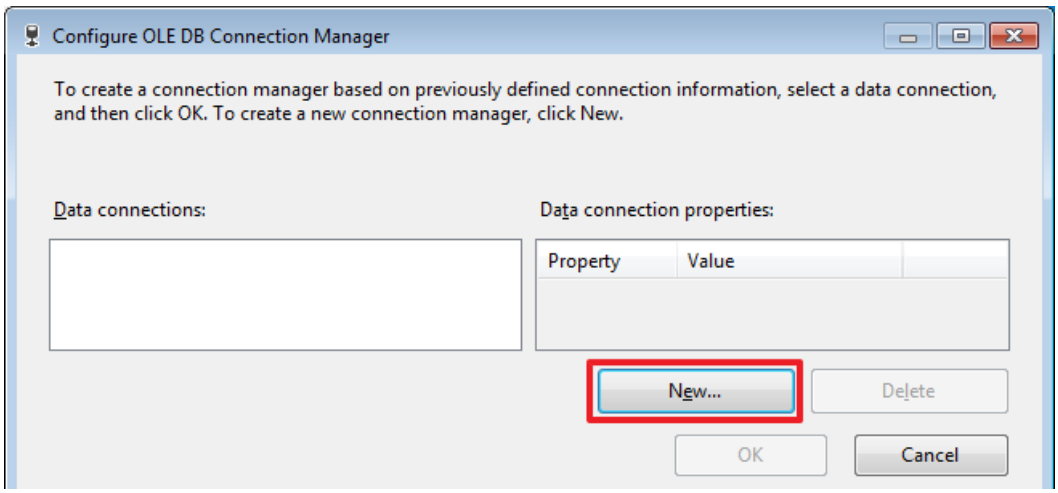
Те ж саме зробимо в властивості пакета, який створився за замовчуванням:



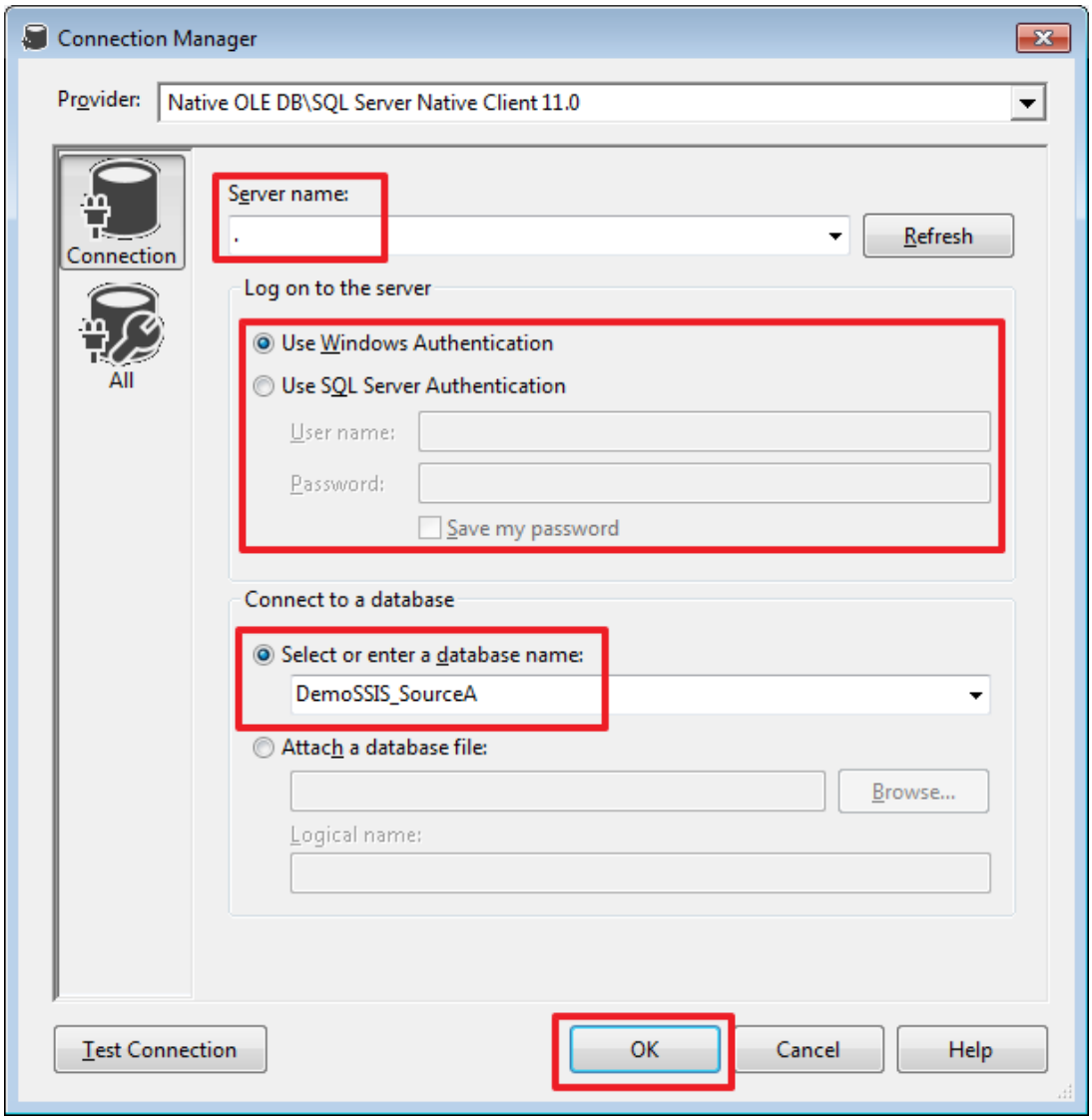
Для всіх нових пакетів ця властивість буде заповнюватися значенням з властивості проекту.
Створимо з'єднання:



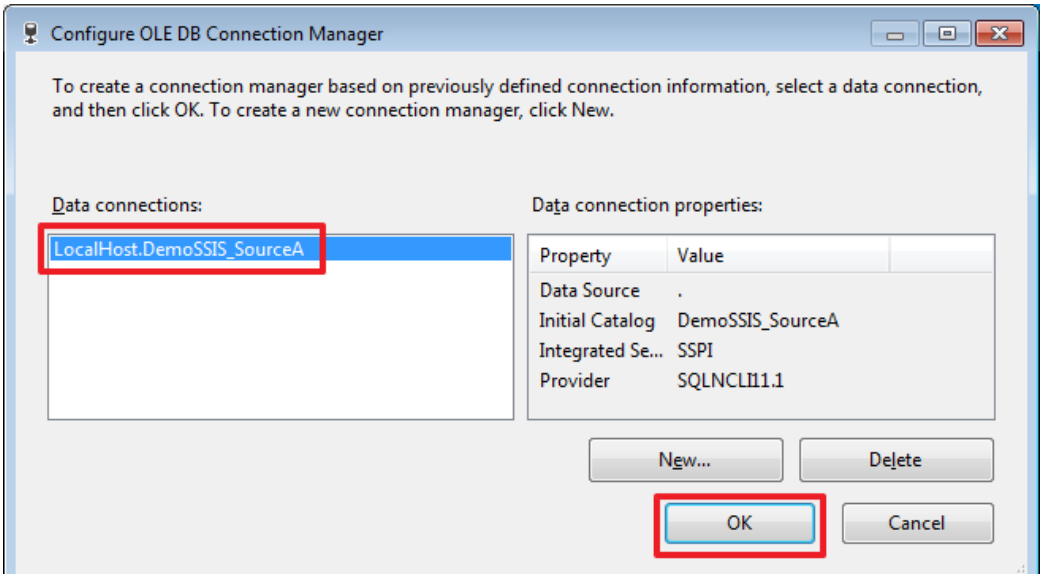




Заповнюємо параметри з'єднання з БД:

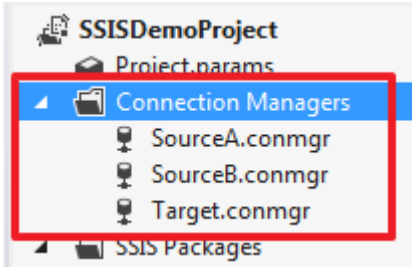


Бойові параметри з'єднання в подальшому можна буде налаштувати при створенні завдання SQL Server Agent.

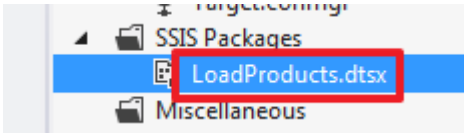


Для зручності я перейменують назву з'єднання на **SourceA**:

Таким же чином створимо і перейменуємо з'єднання для баз **DemoSSIS_SourceB** и **DemoSSIS_Target**:

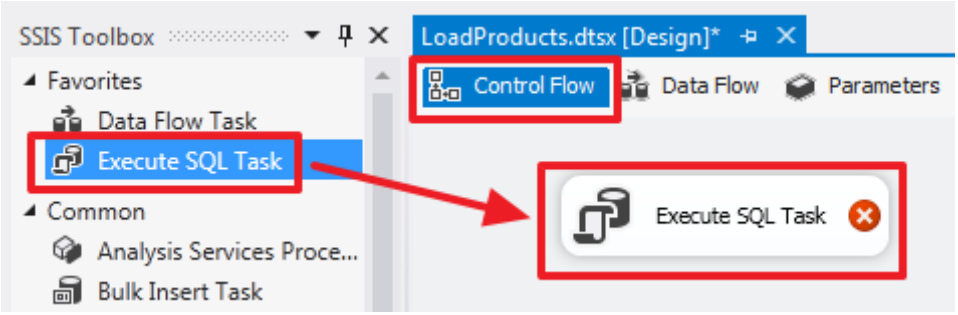


Перейменуємо пакет, створений за замовчуванням, в «**LoadProducts.dtsx**»:

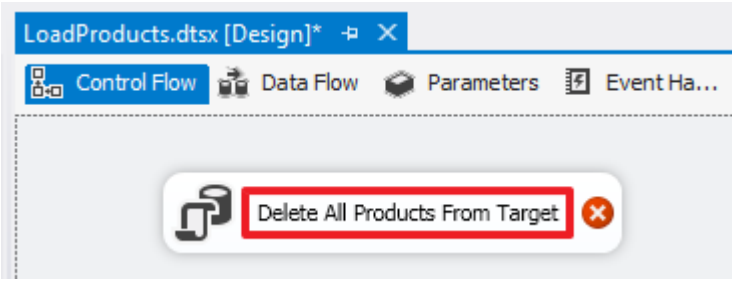


Спочатку напишемо просту логіку, яка буде повністю очищати таблицю Products в базі DemoSSIS_Target і знову завантажувати в неї дані з двох баз даних DemoSSIS_SourceA і DemoSSIS_SourceB.

Для очищення скористаємося компонентом «Execute SQL Task», який ми за допомогою миші створимо в області «Control Flow»:

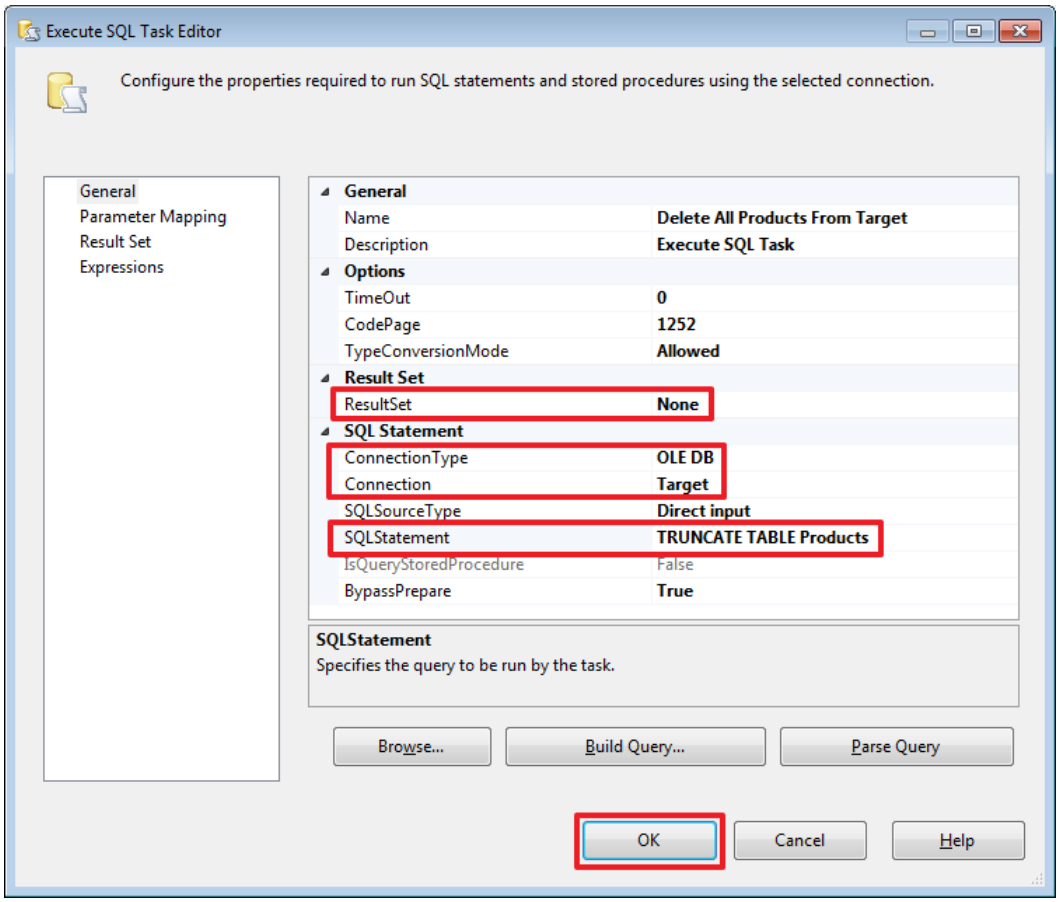


Для наочності можна перейменувати назву компонент. Задамо йому ім'я **«Delete All Products From Target»**:

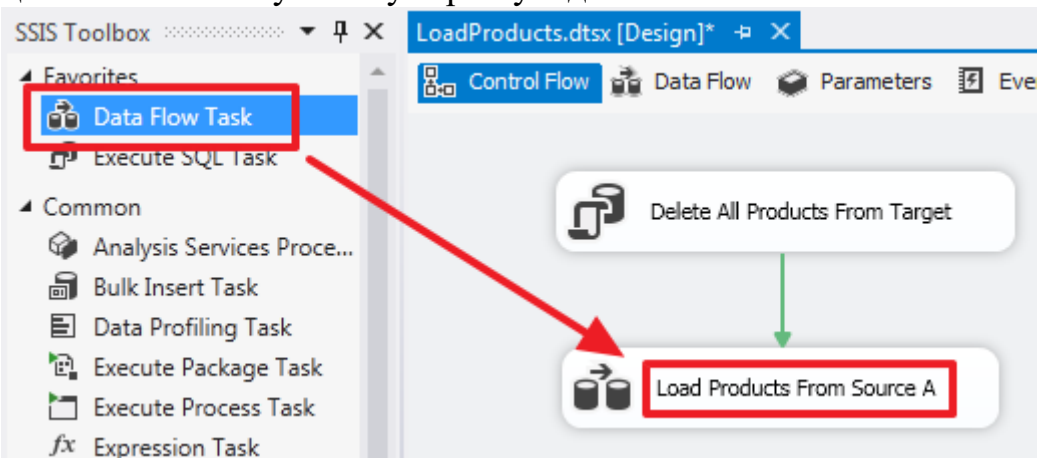


Для цієї мети використовується властивість Name.

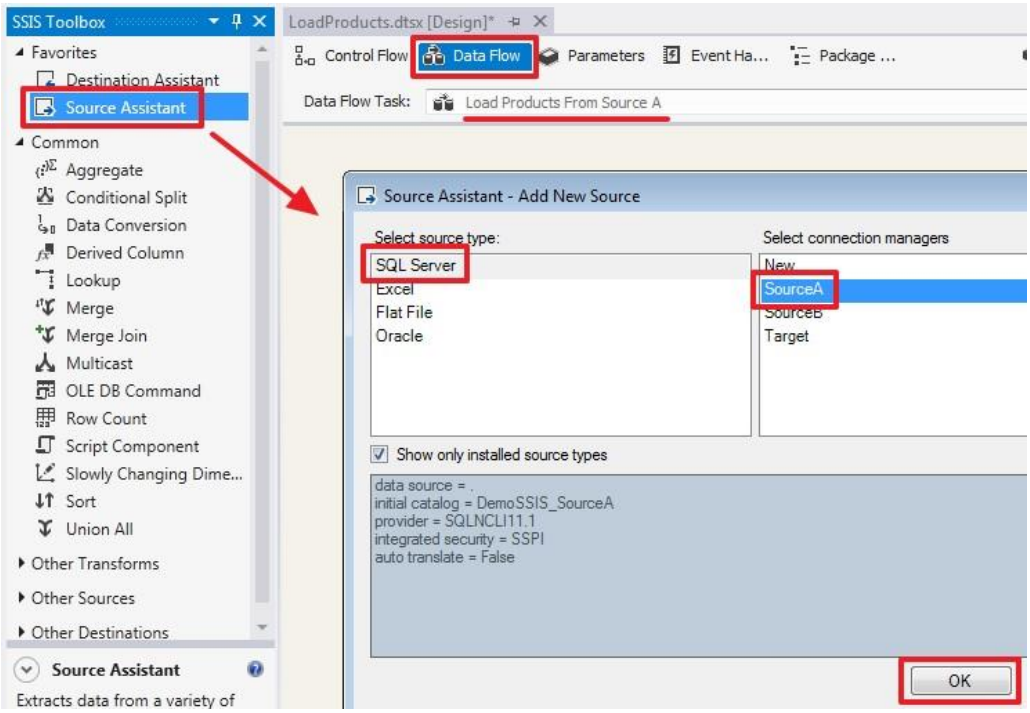
Двічі клацнемо на цьому елементі і пропишемо такі властивості:

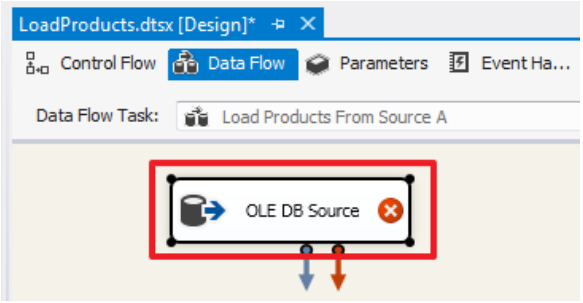


Оскільки TSQL команда «TRUNCATE TABLE Products» нічого не повертає залишимо властивості ResultSet рівним None.
Надалі ми розглянемо, як користуватися параметрами і яким чином можна скористатися результатом виконання команди, записаної в SQLStatement, а поки спробуємо побачити всю картину як це працює в цілому.
Тепер скинемо в область «Control Flow» компонент «Data Flow Task» і перейменуємо його у «Load Products From Source A», а також протягнемо до цього компоненту зелену стрілку від «Delete All Products From Target»:

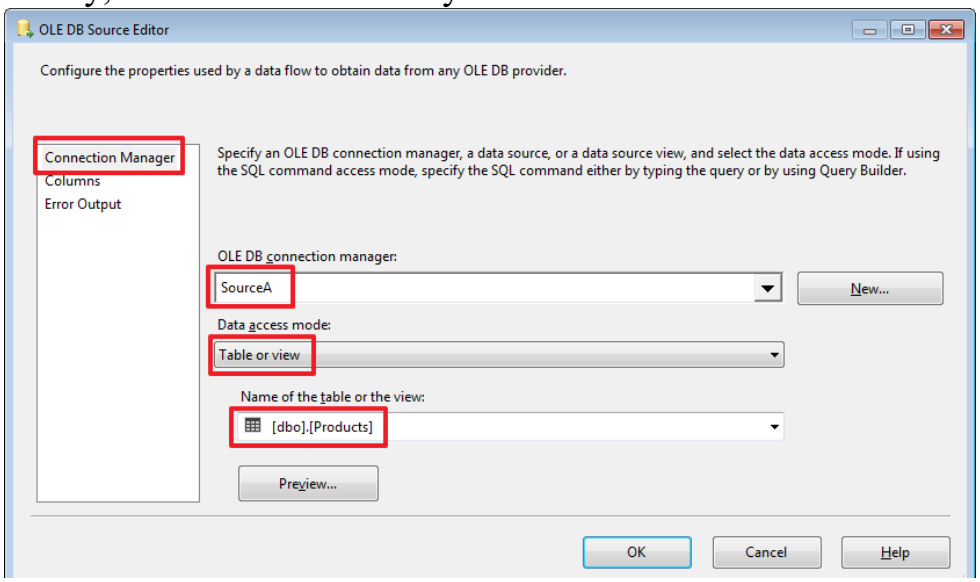


Таким чином ми створили ланцюжок, яка буде виконуватися послідовно.
Клацнувши двічі на «Load Products From Source A» ми потрапляємо в область «Data Flow» цього елемента.
Data Flow Task - це складний компонент, який має свою область, в якій створюються вкладені елементи для роботи з потоком даних.
Скинемо в цю область компонент «Source Assistant»:

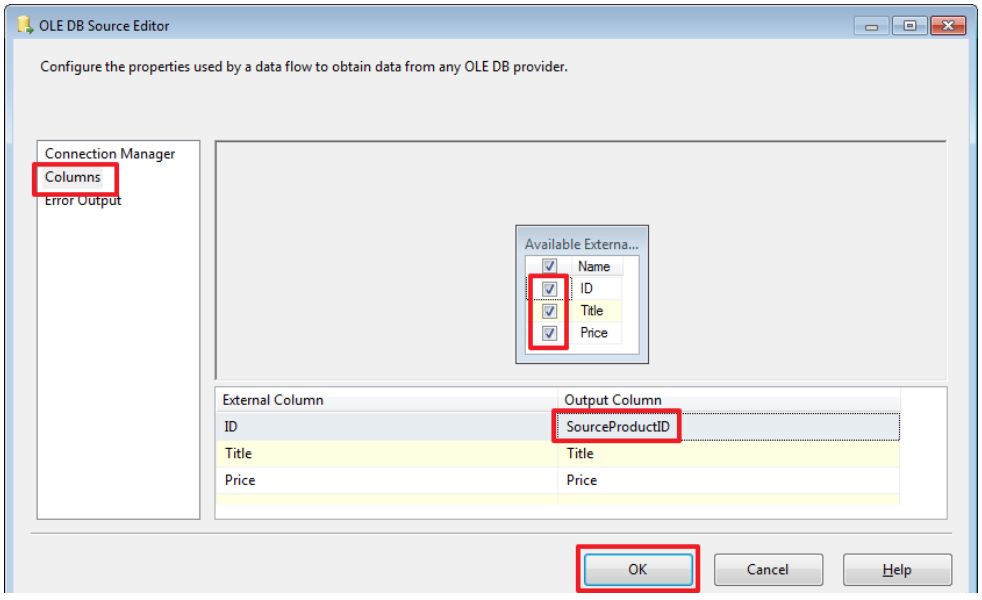




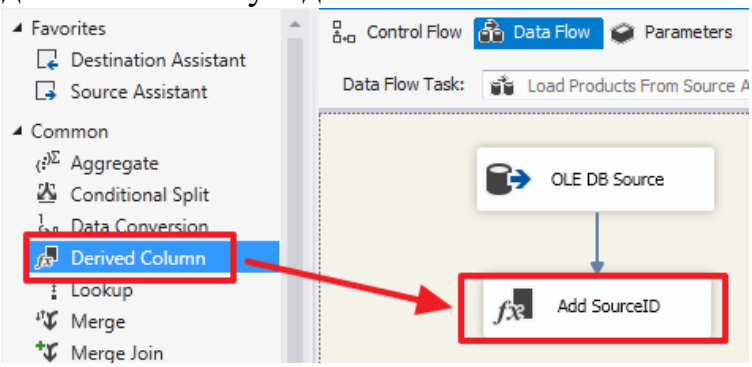
Цей компонент відповідає за отримання даних з джерела. Двічі клацнувши по ньому, ми зможемо налаштувати його:



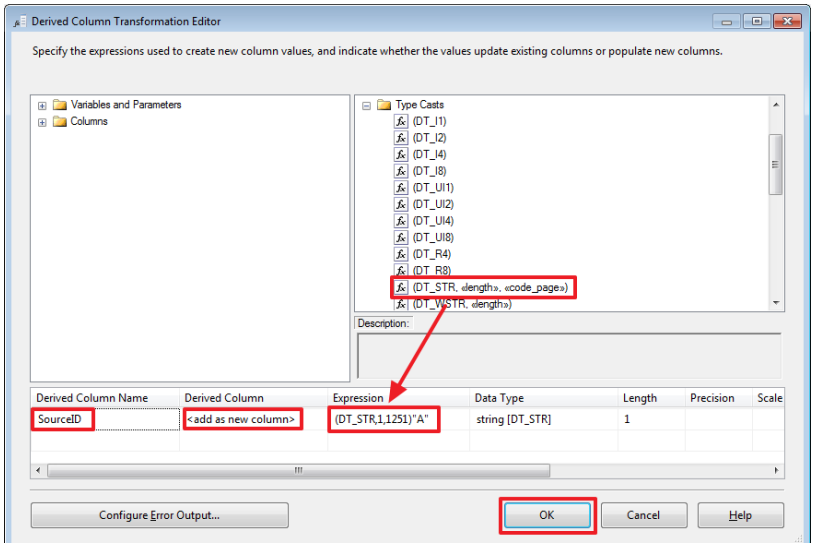
Поки скористаємося режимом «Data access mode» рівним «Table or view». Це призведе до отримання всіх рядків з таблиці Products. Подивитися дані можна натиснувши на «Preview ...».
На закладці Columns ми можемо вибрати тільки необхідні нам колонки і при необхідності перейменувати їх прописавши нове ім'я в колонці «Output Columns»:



Для одержувача потрібна ще одна додаткова колонка SourceID, додамо її до вихідного набору за допомогою компонента «Derived Column», який перейменуємо в «Add SourceID», так само протягнемо синю стрілку до даного елементу від «OLE DB Source»:

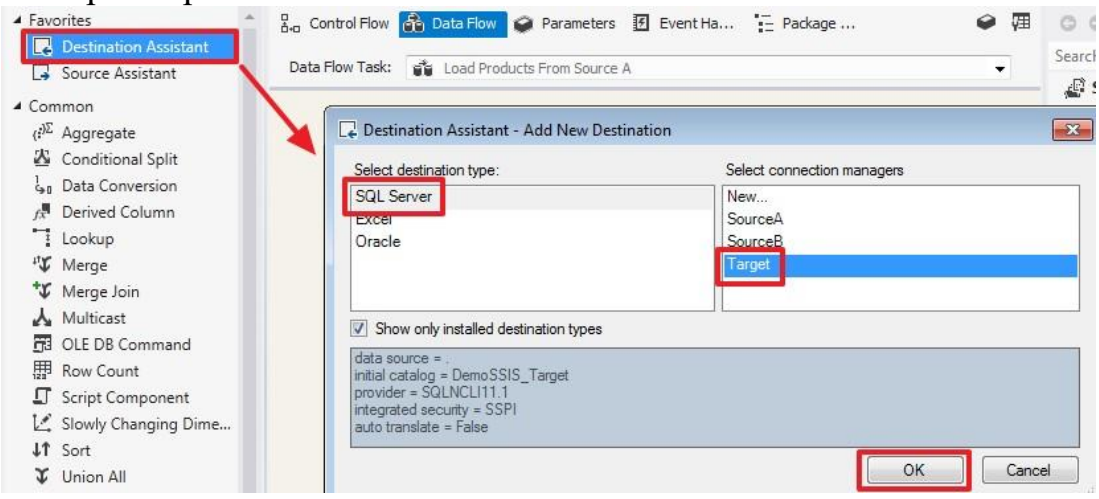


Двічі клацнемо по елементу «Add SourceID» і пропишемо значення «A» у вигляді константи:

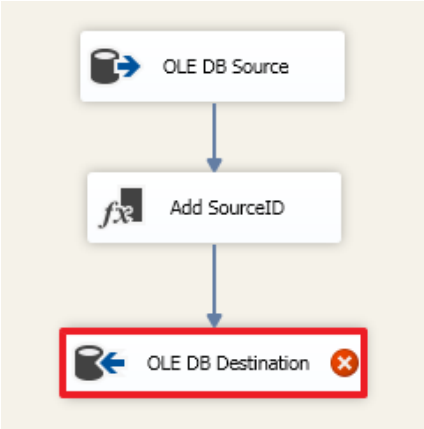


Тут я скористався функцією перетворення типу (DT_STR, 1,1251) для того щоб перетворити Unicode рядок в ANSI.

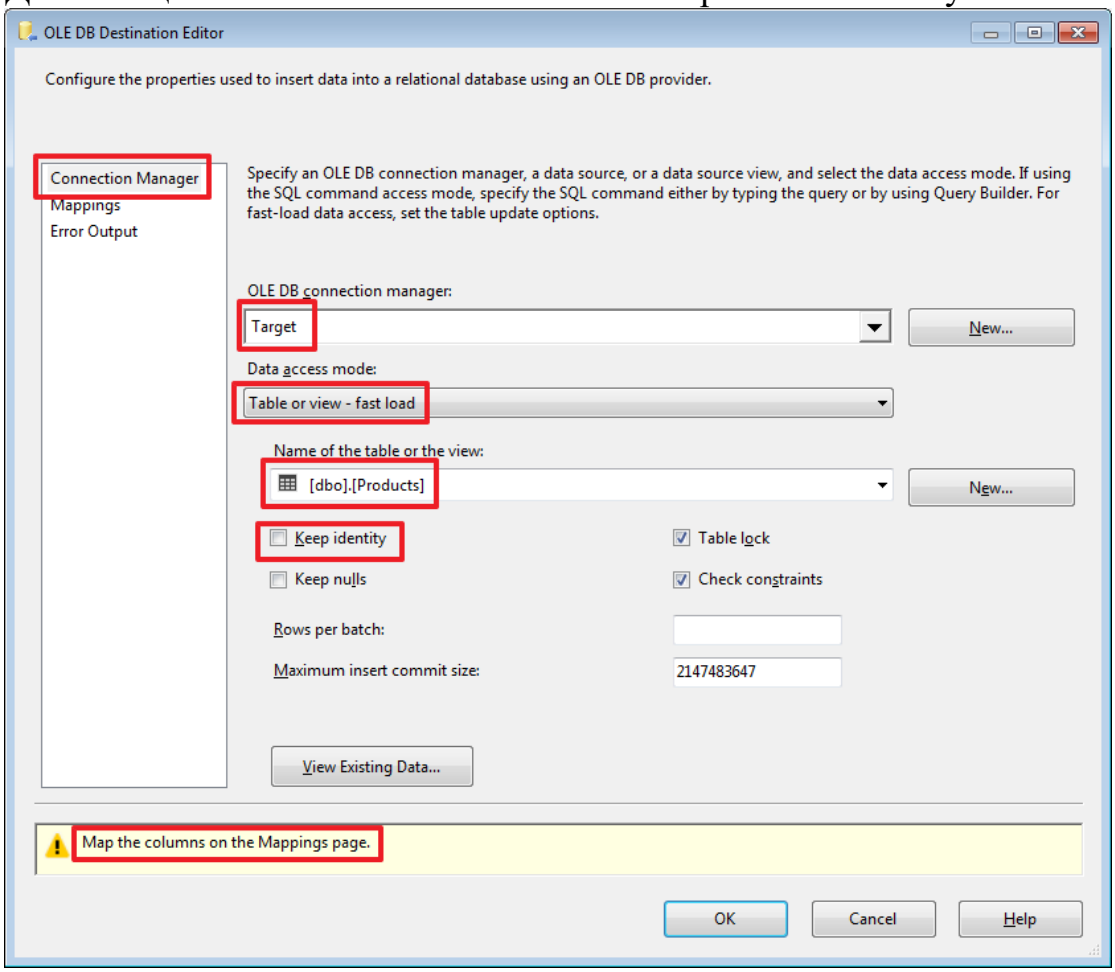
Тепер створимо компонент «Destination Assistant»:



Направимо в нього потік від «Add SourceID»:



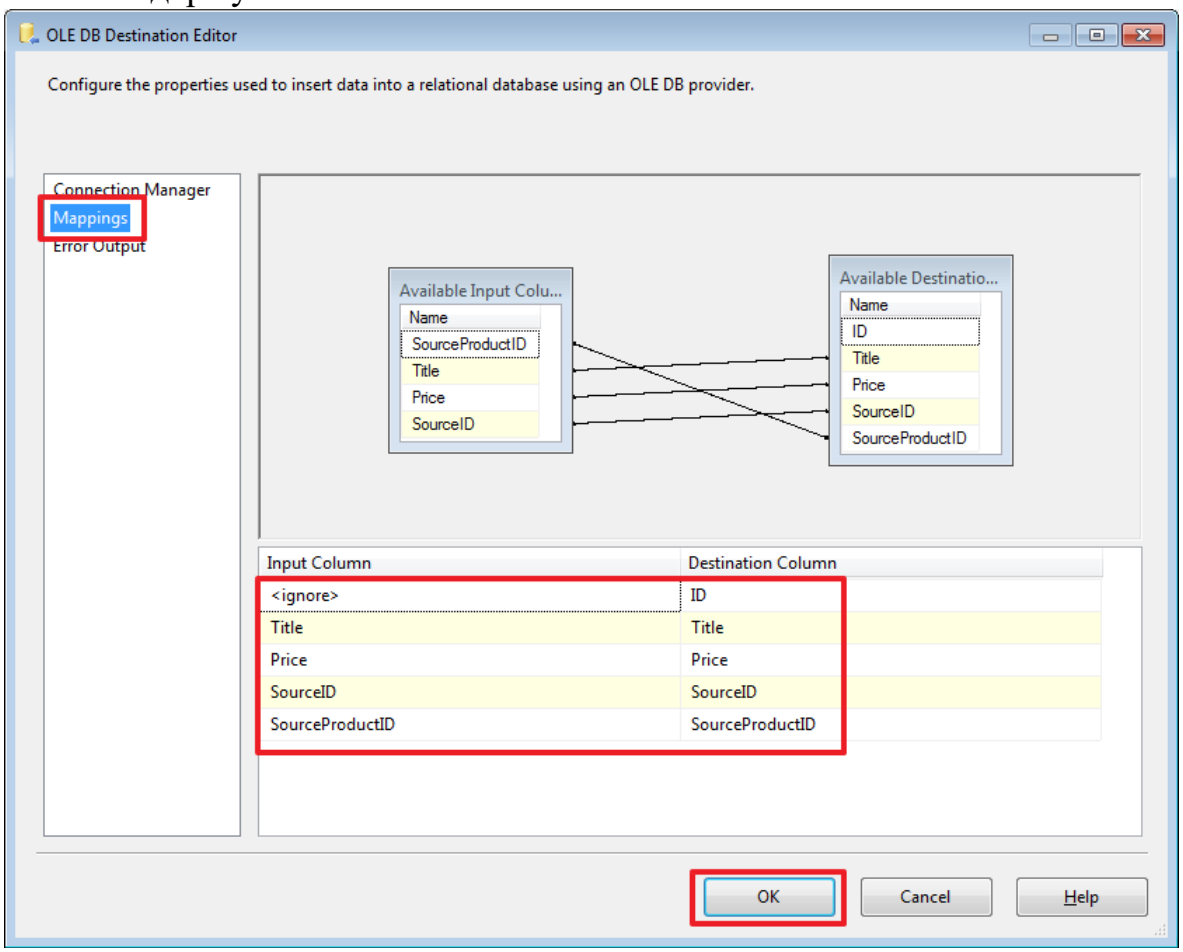
Двічі клацнемо по «OLE DB Destination» і зробимо налаштування



Тут ми показуємо в яку таблицю буде записуватися отриманий набір.

«Keep identity» використовується в разі якщо в приймаючій таблиці є поле з прапором IDENTITY і ми хочемо, щоб значення в нього теж записувалися з джерела (це аналогічно включенню опції SET IDENTITY_INSERT Products ON).

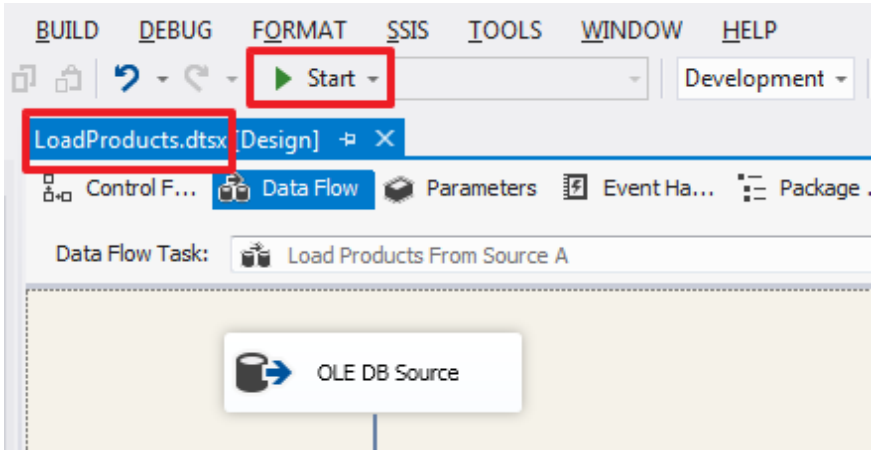
Перейшовши на закладку Mappings здійснимо прив'язку полів джерела з полями одержувача:



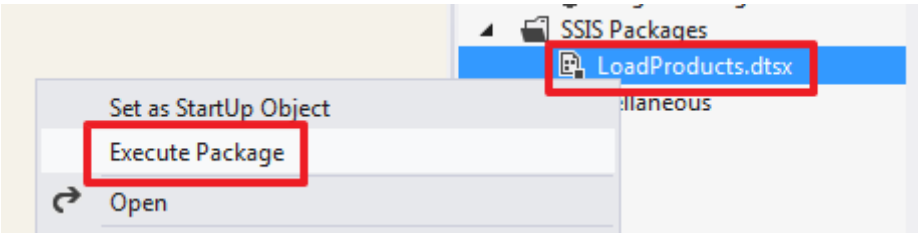
Так як у нас поля джерела і приймача іменуються однаково, то прив'язка здійснилася автоматично.

Чи можемо протестувати роботу пакета і переконатися, що дані залились в таблицю Products бази DemoSSIS_Target.

Запускаємо пакет на виконання з Visual Studio натиснувши **Start** або клавішу **F5**:

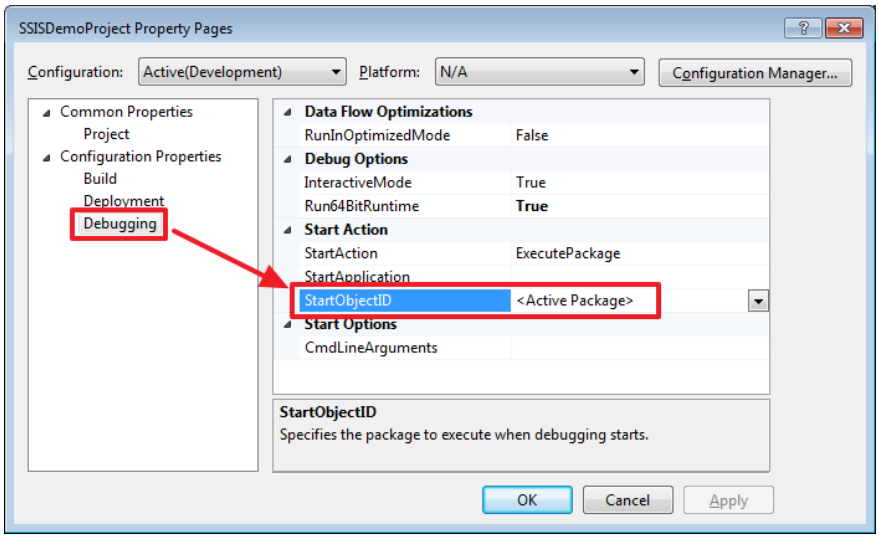


Так само пакет можна виконати, скориставшись командою з контекстного меню:



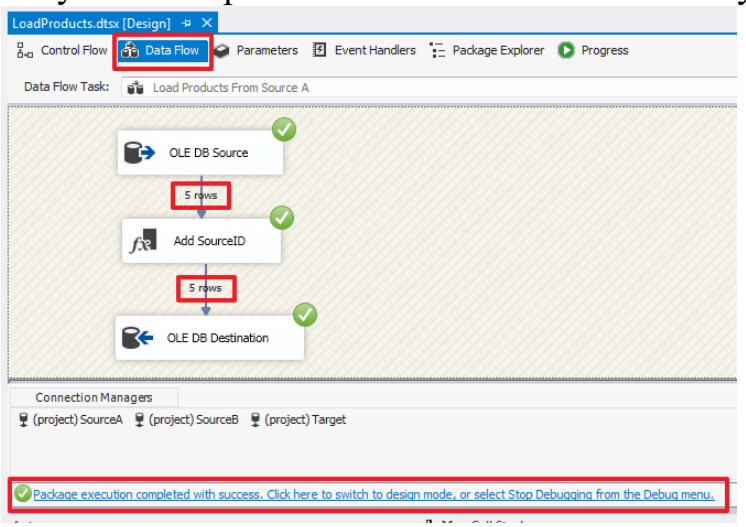
За допомогою «**Set as StartUp Object**» можна задати пакет, який буде запускатися після натискання на **Start (F5)**.

Який пакет буде запускатися при натисканні на **Start (F5)** можна перевизначити у властивостях проекту:



За замовчуванням буде запускатися пакет відкритий в поточний момент, про це говорить значення StartObjectID рівне <Active Package>.

Запустивши проект ми повинні побачити наступну картину:



Пакет виконався без помилок, про що говорить зелений значок і текст в нижній частині.

У разі наявності помилок їх можна буде побачити вкладці Progress.

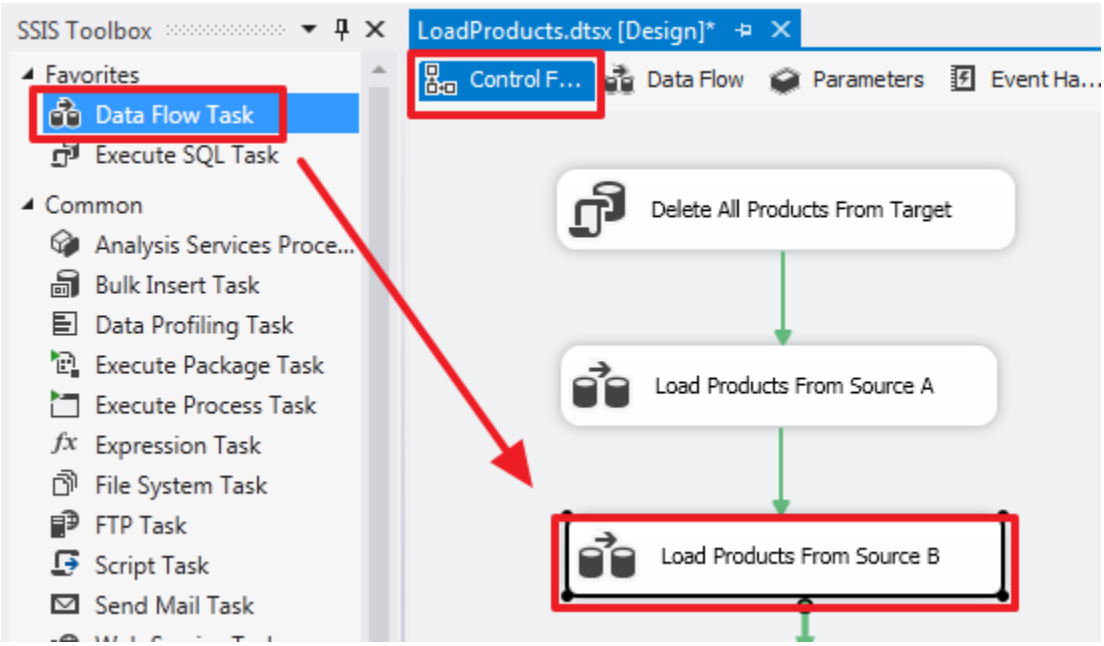
Натиснемо на посилання «Package execution completed ...» або на кнопку «Stop Debugging» розташовану на панелі інструментів для зупинення виконання пакету.

Виконуємо запит:

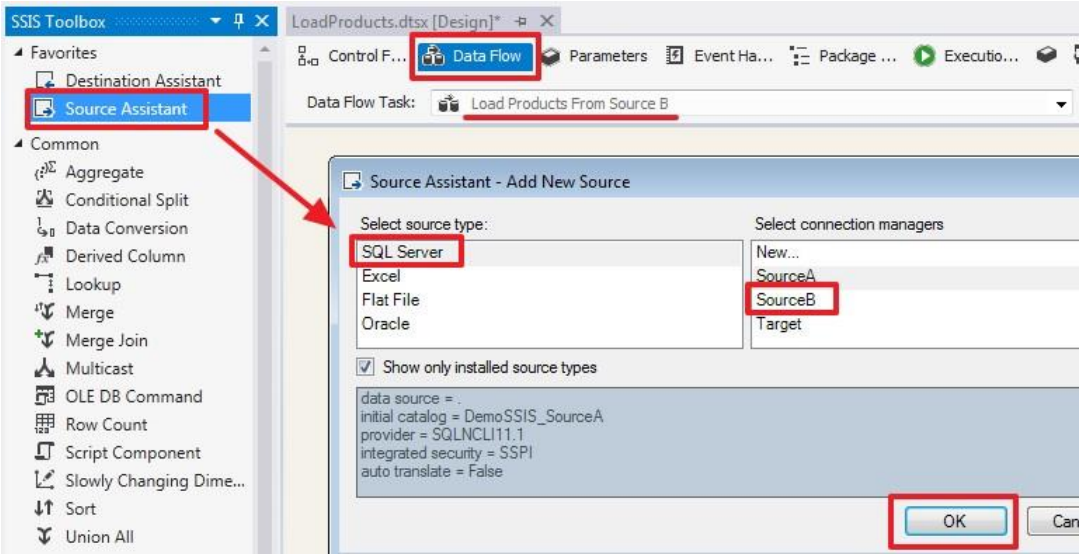
```
USE DemoSSIS_Target
GO

SELECT *
FROM Products
```

І переконаємося, що дані були записані в приймаючу таблицю.

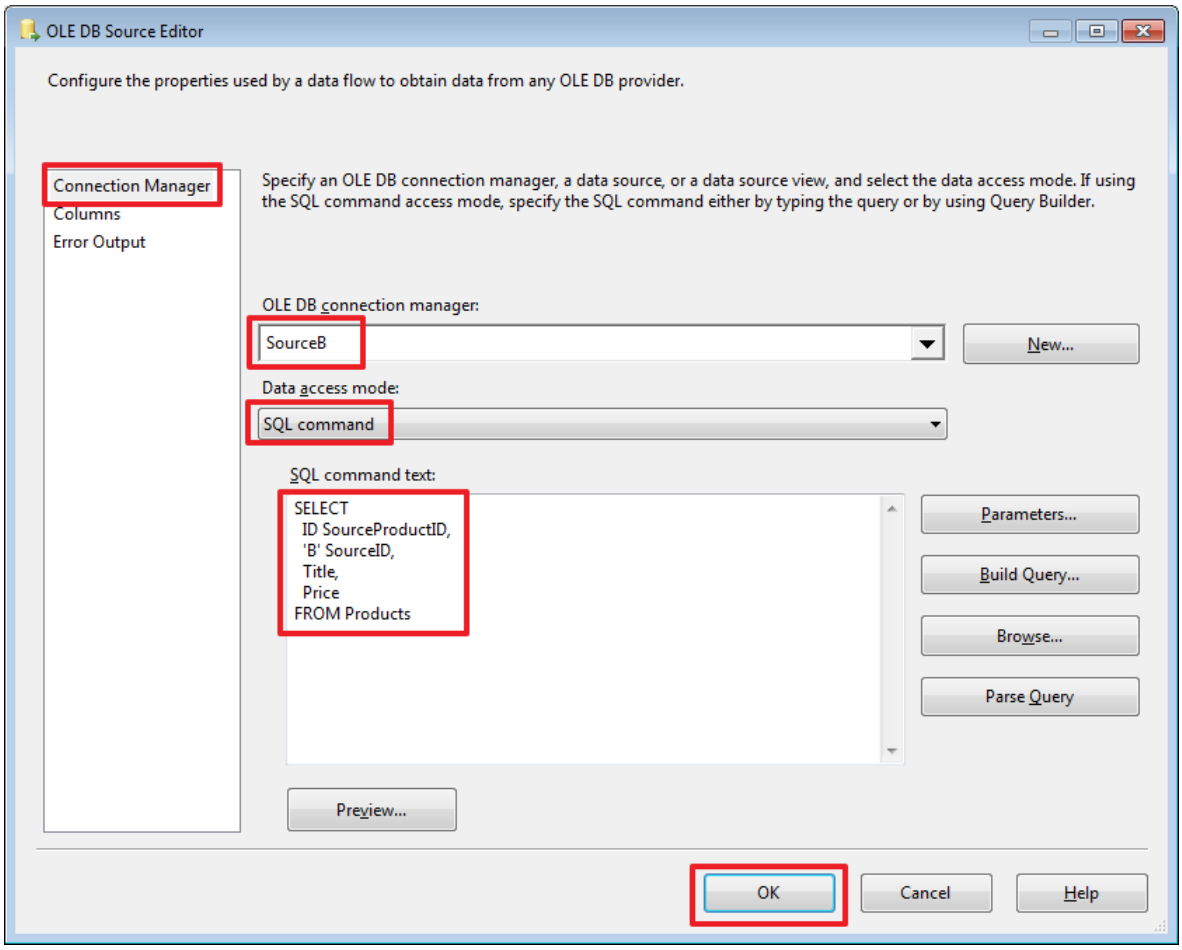
Перейдемо в область «Control Flow» і створимо ще один компонент «Data Task Flow», який назвемо «Load Products From Source B», протягнемо на нього зелену стрілку від «Load Products From Source A»:



Подвійним клацанням зайдемо в область «Data Flow» цього елемента і створимо «Source Assistant» :



Двічі клацнувши на цьому елементі, налаштуємо його по-іншому:

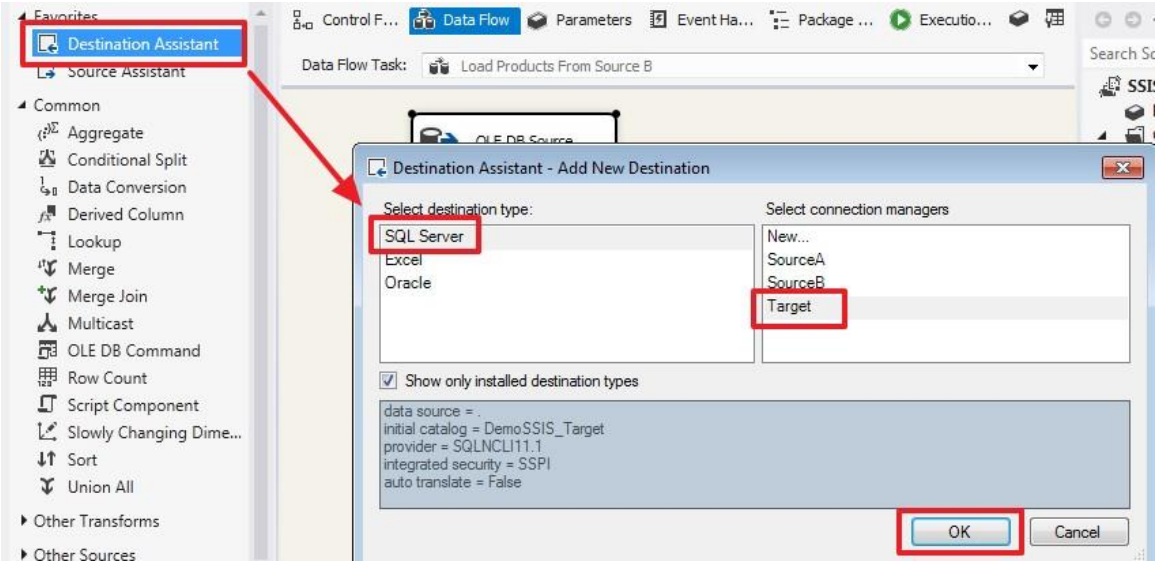


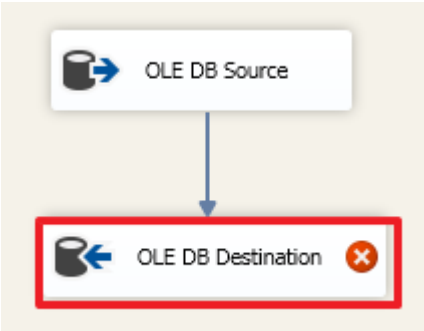
Віберемо режим «SQL command» и пропішемо Наступний запит

```
SELECT
ID SourceProductID,
'B' SourceID,
 Title,
```

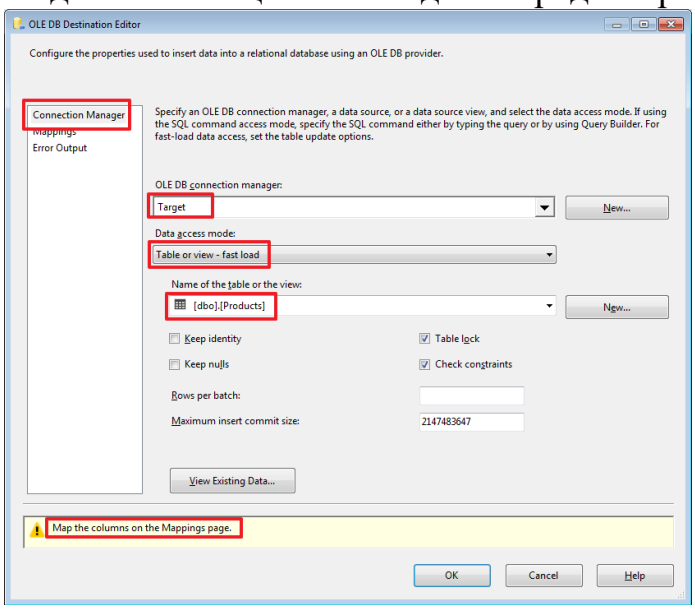
```
Price
FROM Products
```

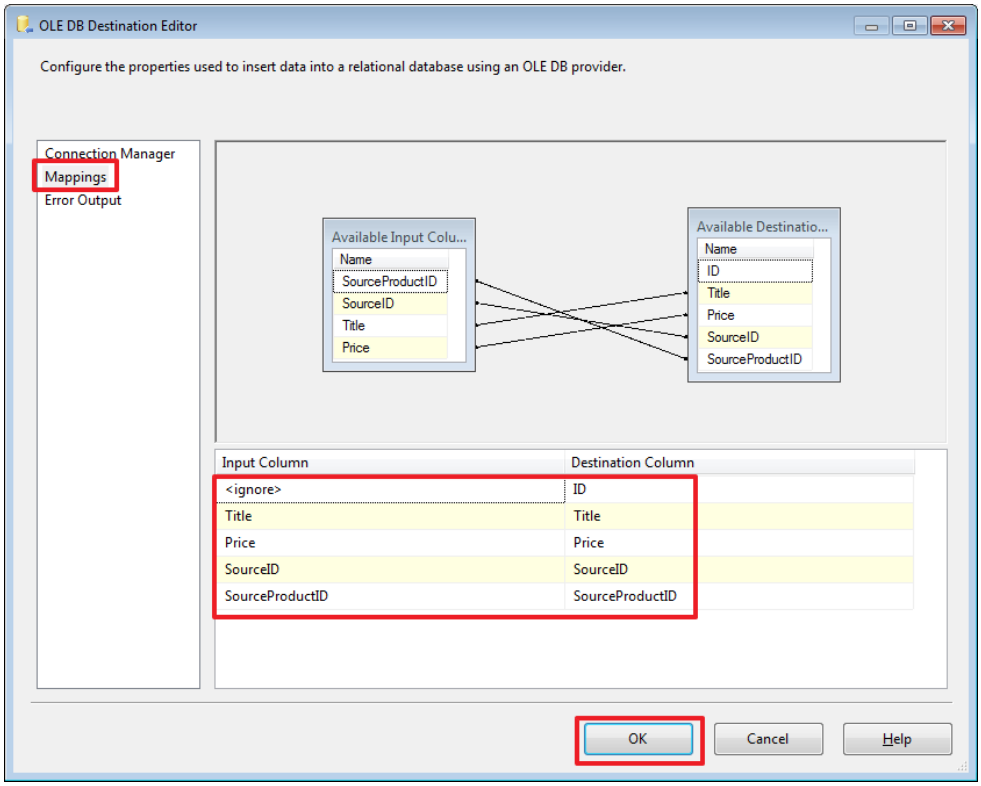
Далі відразу створімо компонент «Destination Assistant» и протягнемо на нього синю стрілку від «OLE DB Source»:





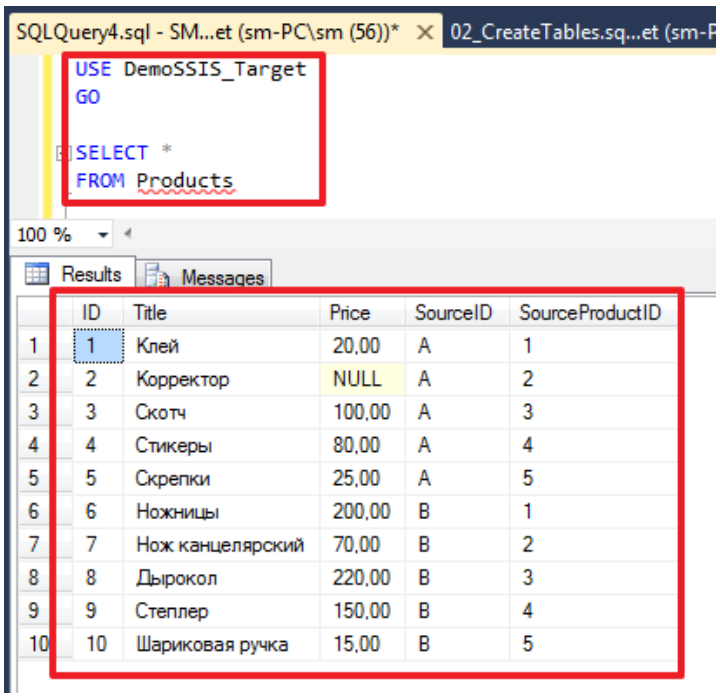
Подвійним клацанням зайдемо в редактор цього елемента і налаштуємо його:





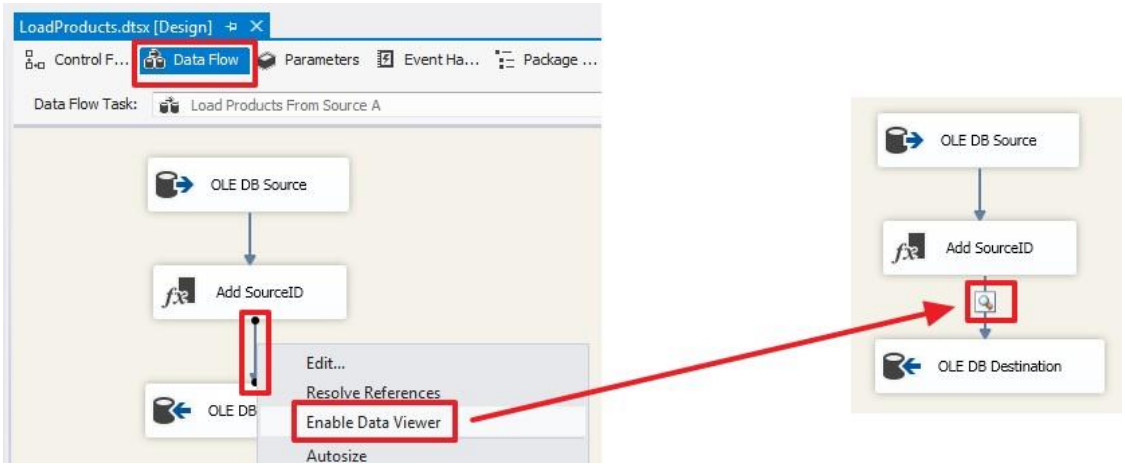
Запустимо проект на виконання і переконаємося, що дані з двох джерел потрапили в таблицю в базі Target:

```
USE DemoSSIS_Target
GO

SELECT *
FROM Products
```

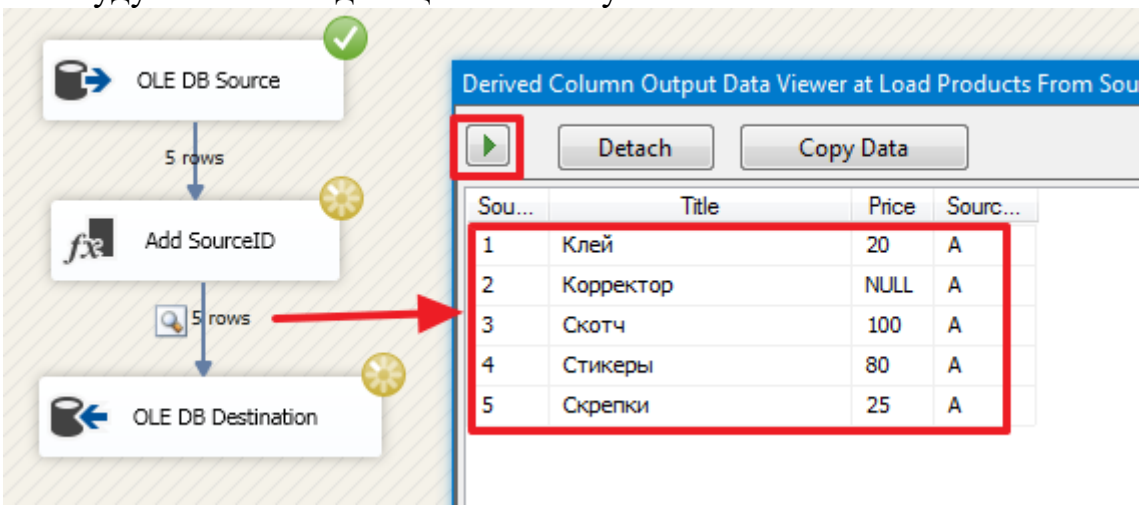


	ID	Title	Price	SourceID	SourceProductID
1	1	Клей	20,00	A	1
2	2	Корректор	NULL	A	2
3	3	Скотч	100,00	A	3
4	4	Стикеры	80,00	A	4
5	5	Скрепки	25,00	A	5
6	6	Ножницы	200,00	B	1
7	7	Нож канцелярский	70,00	B	2
8	8	Дырокол	220,00	B	3
9	9	Степлер	150,00	B	4
10	10	Шариковая ручка	15,00	B	5

Додатково в контекстному меню стрілки можна активізувати «Data Viewer»:

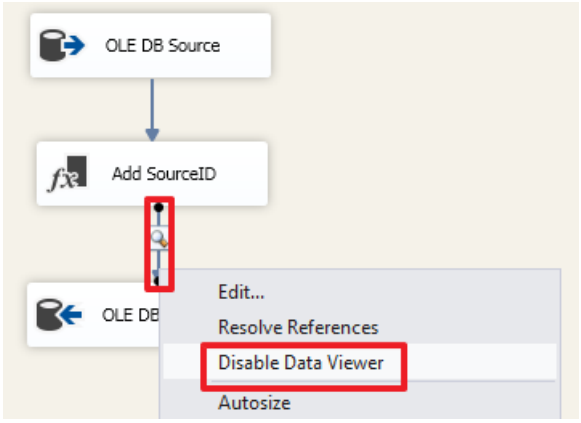


Тепер при запуску пакета на виконання в цій точці буде зроблена зупинка і нам будуть показані дані цього потоку:



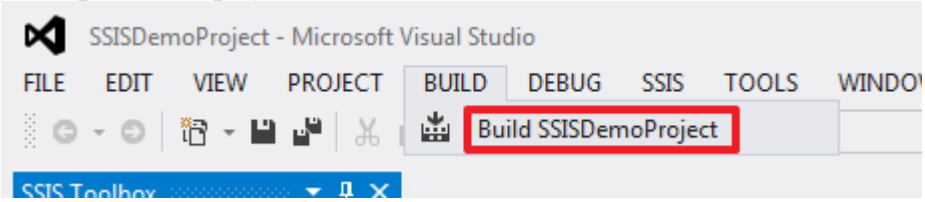
Для продовження виконання пакету потрібно натиснути на кнопку зі стрілкою або просто закрити вікно перегляду даних.

Для відключення цієї функції в контекстному меню стрілки вибираємо «Disable Date Viewer»:



Для першої частини думаю цього буде достатньо.

Створимо збірку:



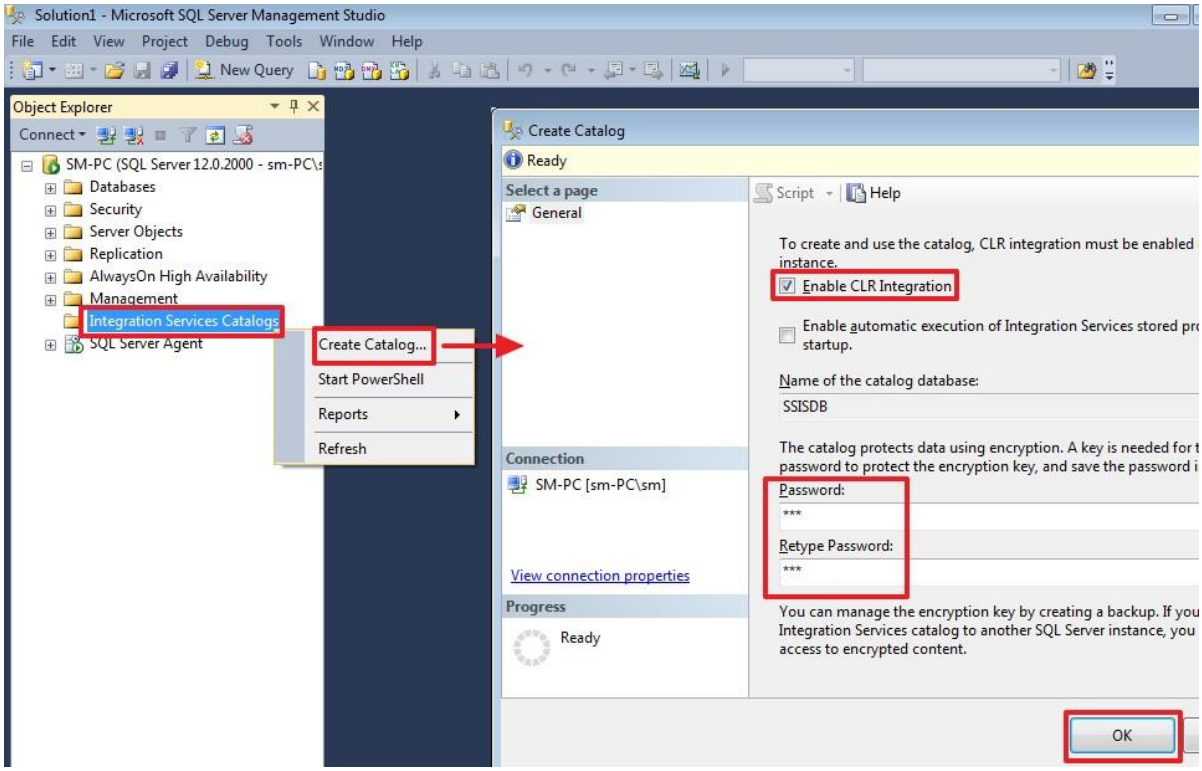
В результаті ми отримаємо файл «C:\SSIS\SSISDemoProject\bin\Development**SSISDemoProject.ispac**».

Розглянемо яким чином робиться розгортання цього проекту на SQL Server.

Розгортання SSIS

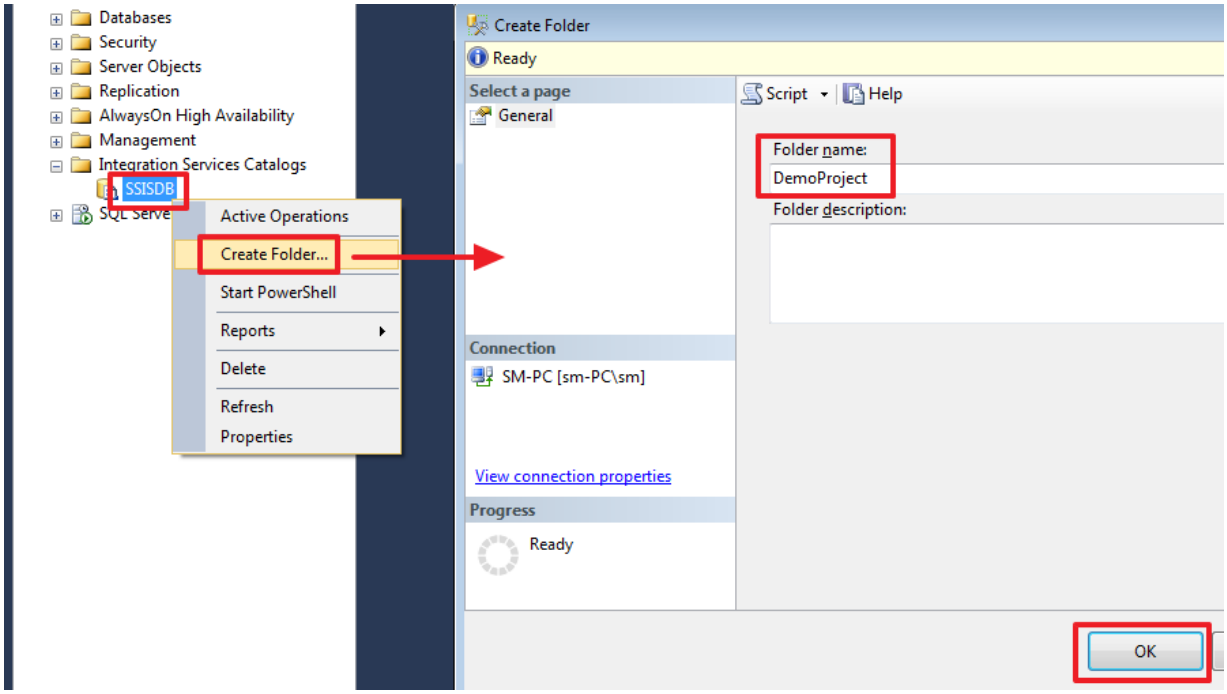
Всі подальші дії будемо робити в SSMS.

Створення каталогу SSISDB:

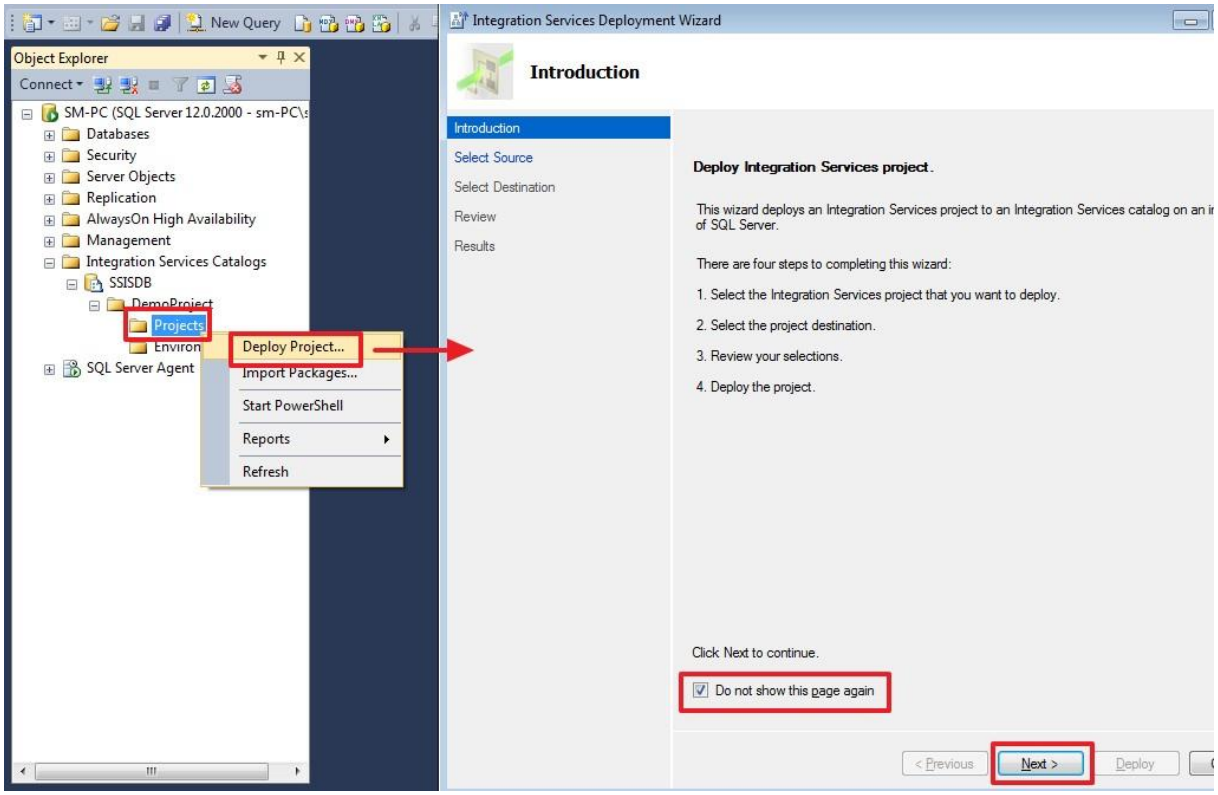


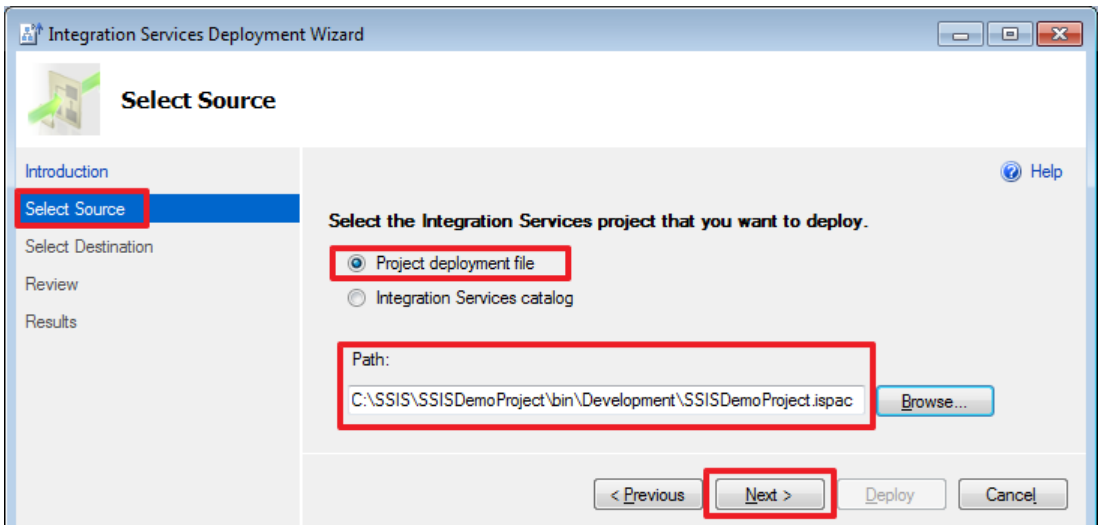
Тут вводимо будь-пароль.

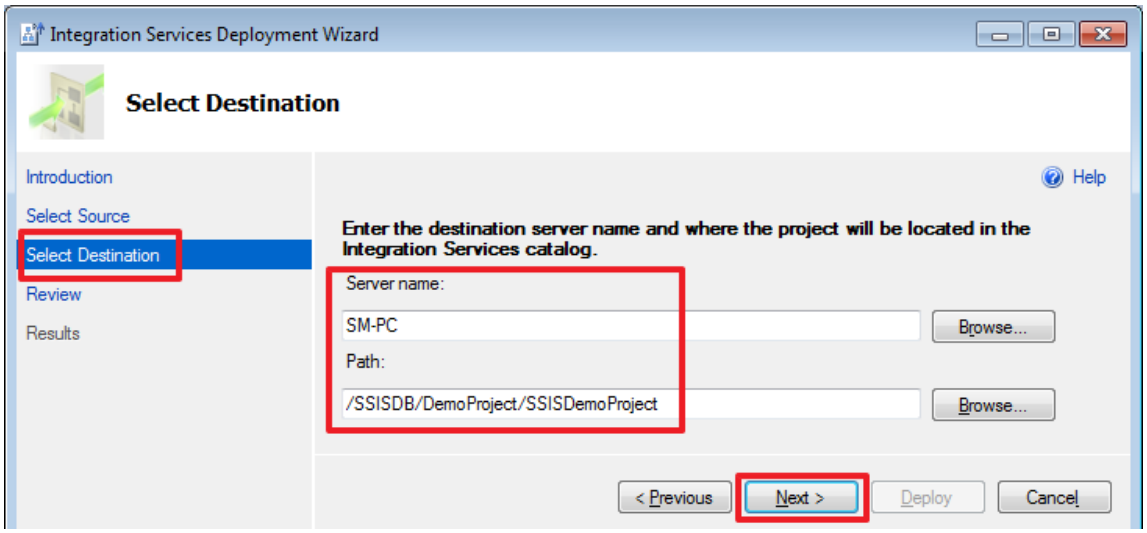
Тепер створюємо папку, в якій буде розташовуватися наш проект:

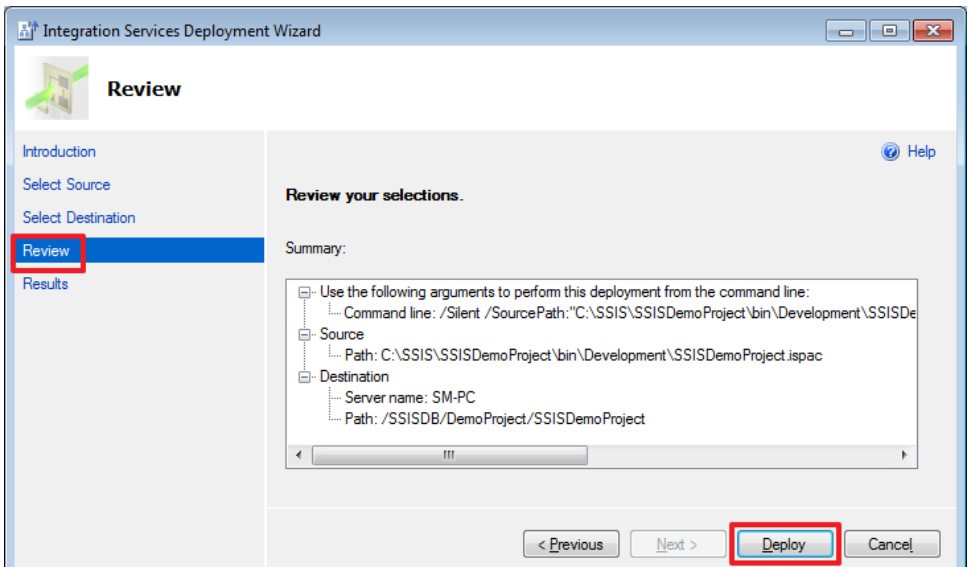


Розгортаємо сам проект:

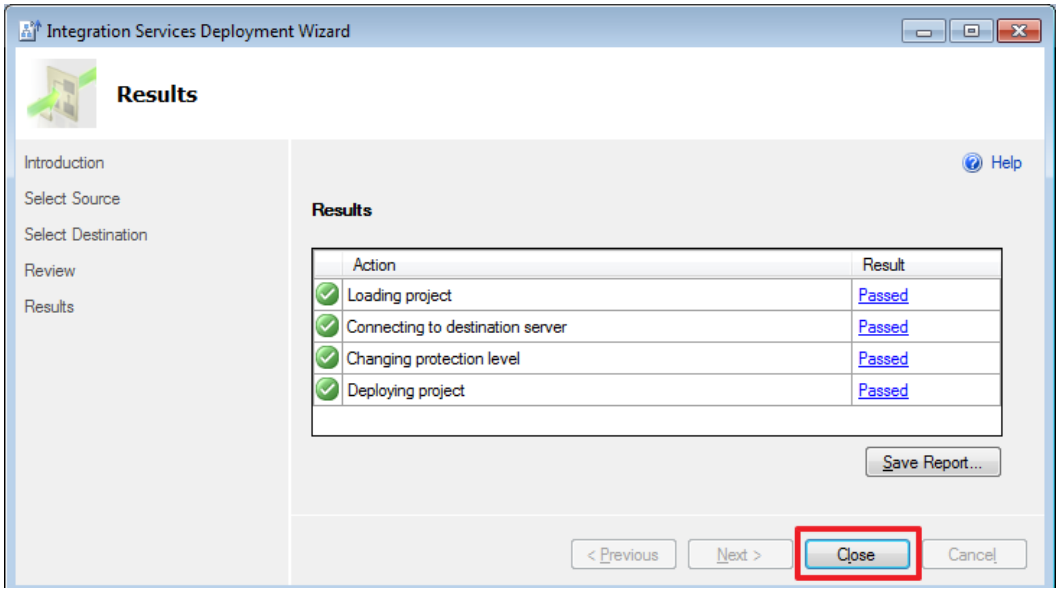




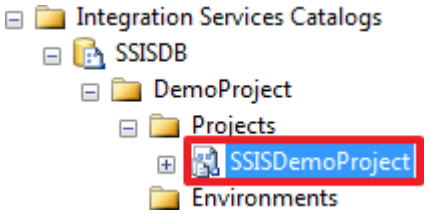




На завершення ми повинні побачити наступну картину:

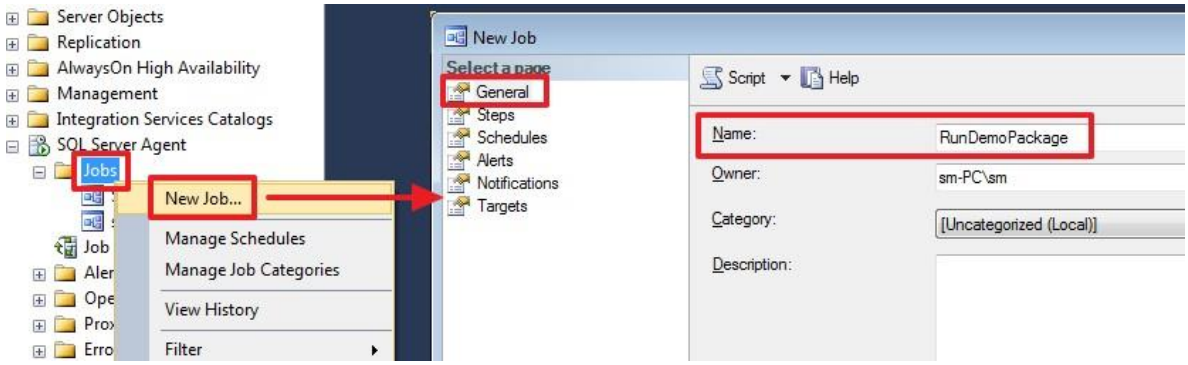


Після поновлення (F5) ми побачимо наш проект:

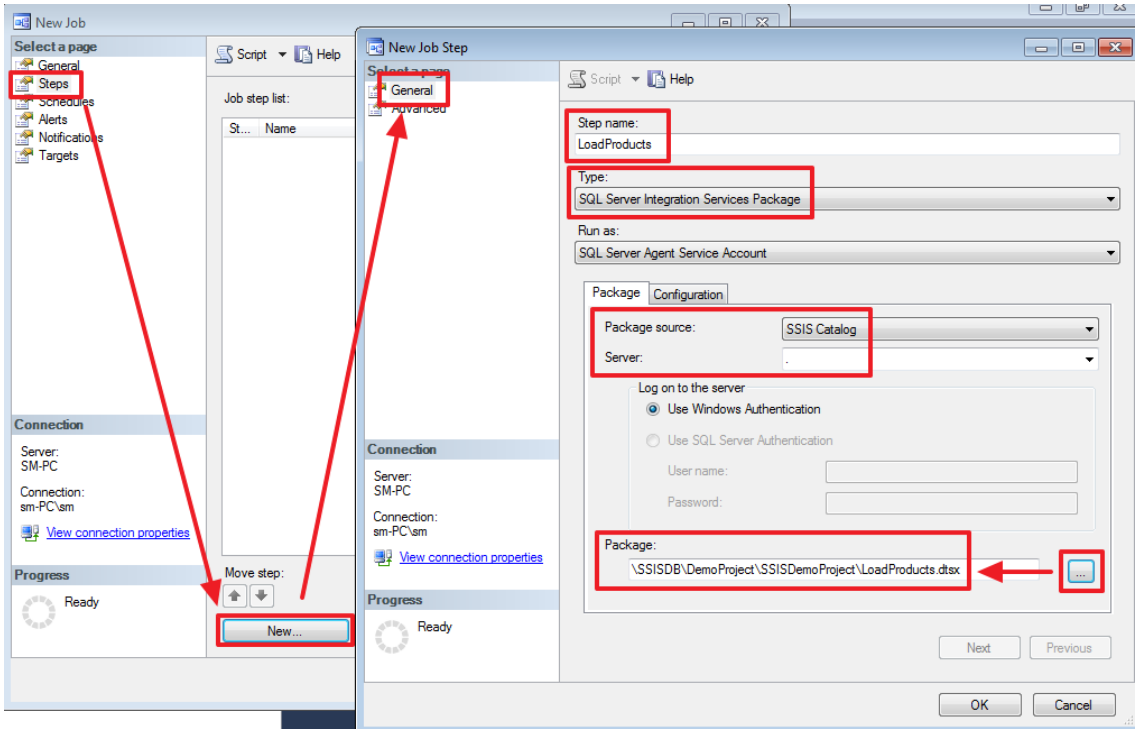


Створення завдання в SQL Server Agent

Створимо завдання в SQL Agent, для виконання пакету за розкладом:

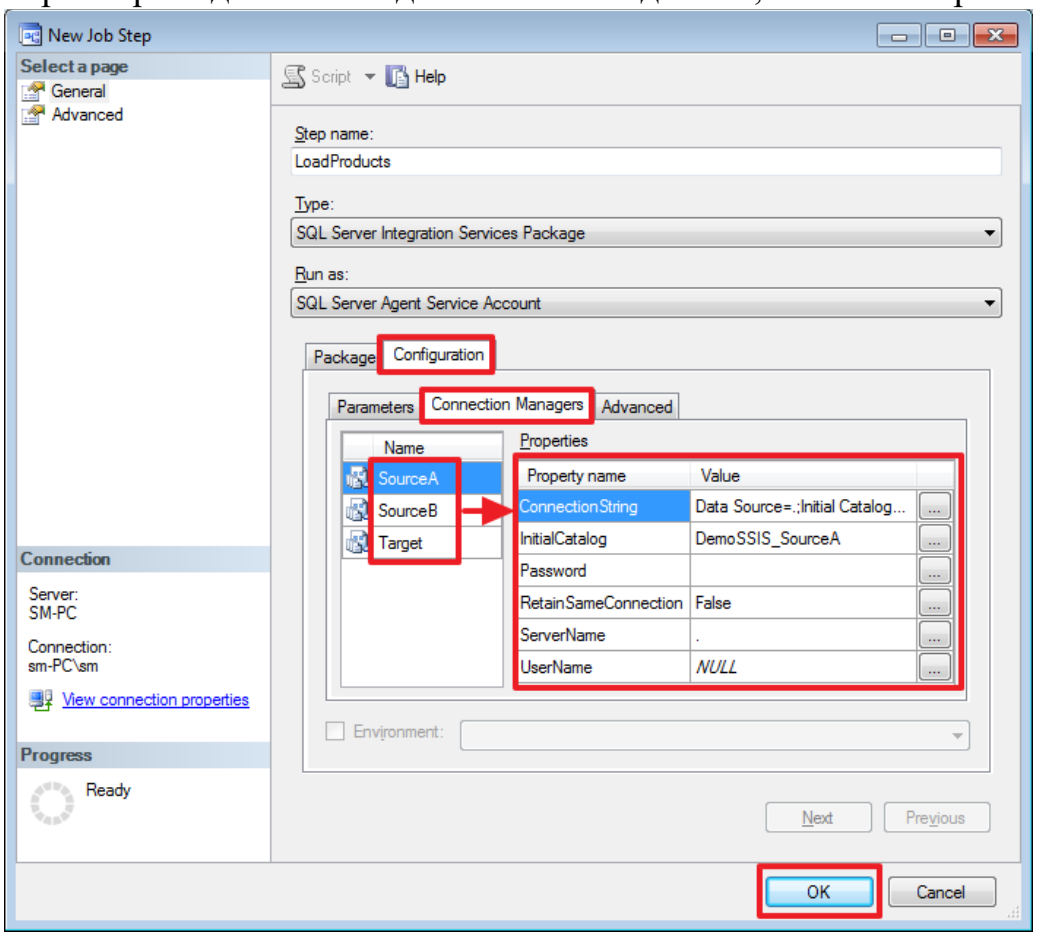


Создаем новый шаг:

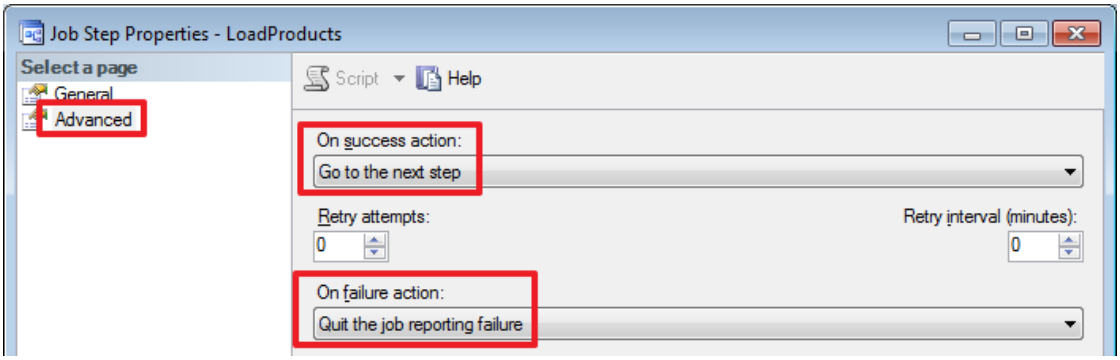


На вкладці «**Configuration → Parameters**» можна задати параметри пакета (їх розглянемо в наступних частинах).

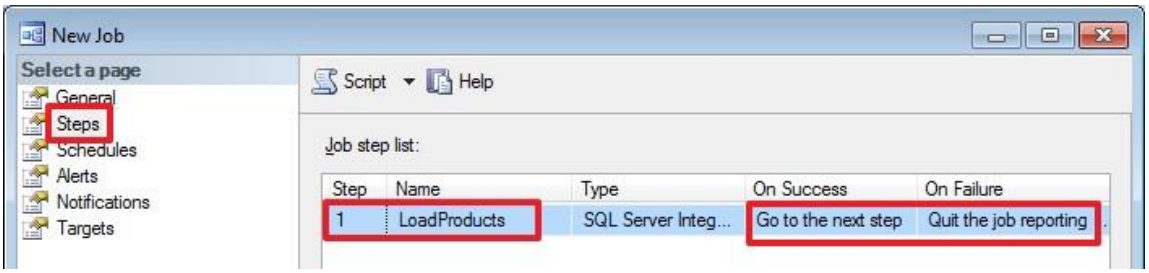
На вкладці «**Configuration → Connection Manager**» ми можемо змінити параметри підключення для кожного з'єднання, яке ми створили в проекті:



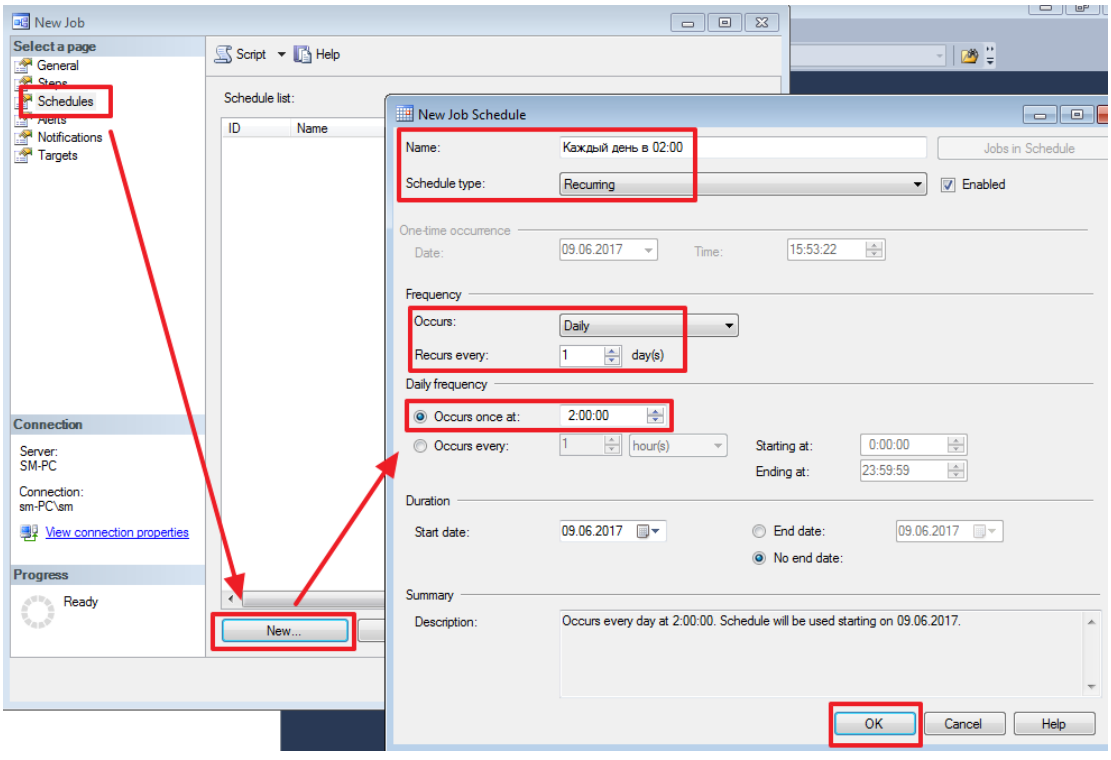
На закладці Advanced можна змінити логіку, яка буде використовуватися при успішному або неуспішному завершення кроку:



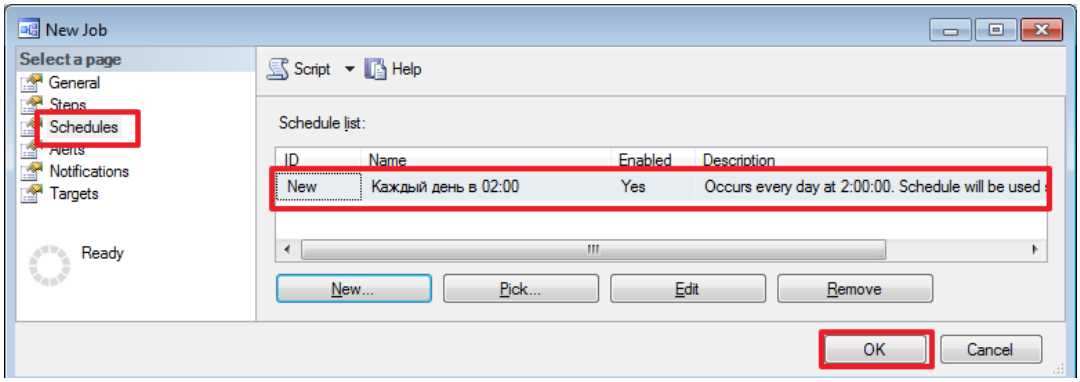
Крок створено:



Залишилося створити розклад для даного завдання:

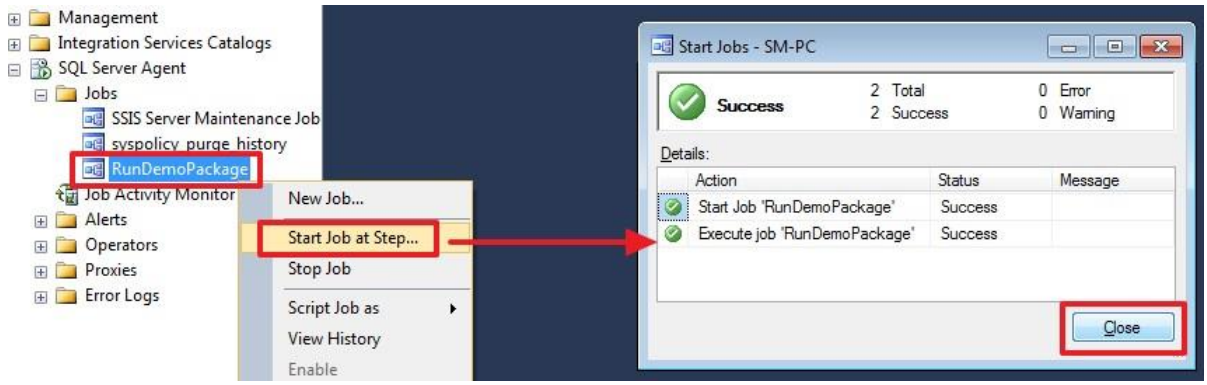


Розклад можна задати різноманітним чином. Думаю, тут все повинно бути інтуїтивно зрозуміло:



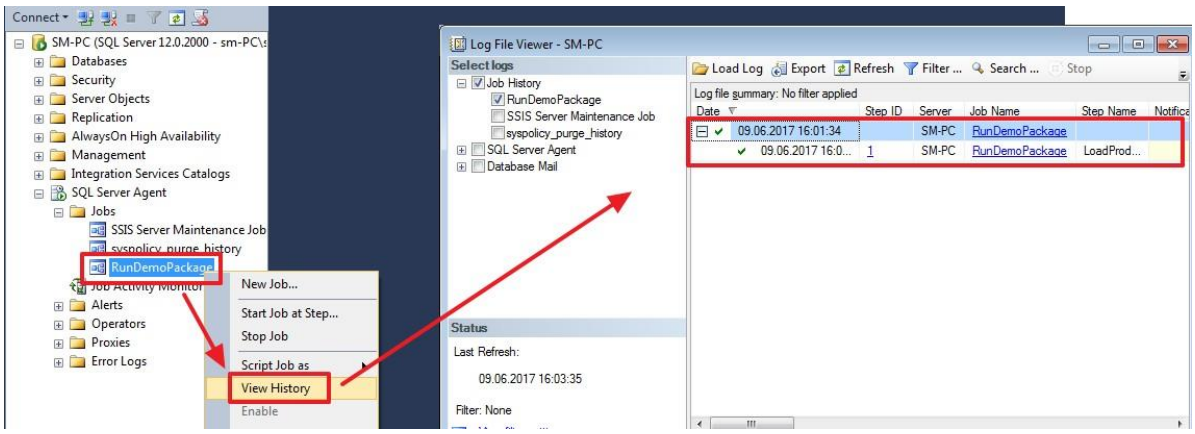
Все, задача створена.

Робимо тестовий запуск:



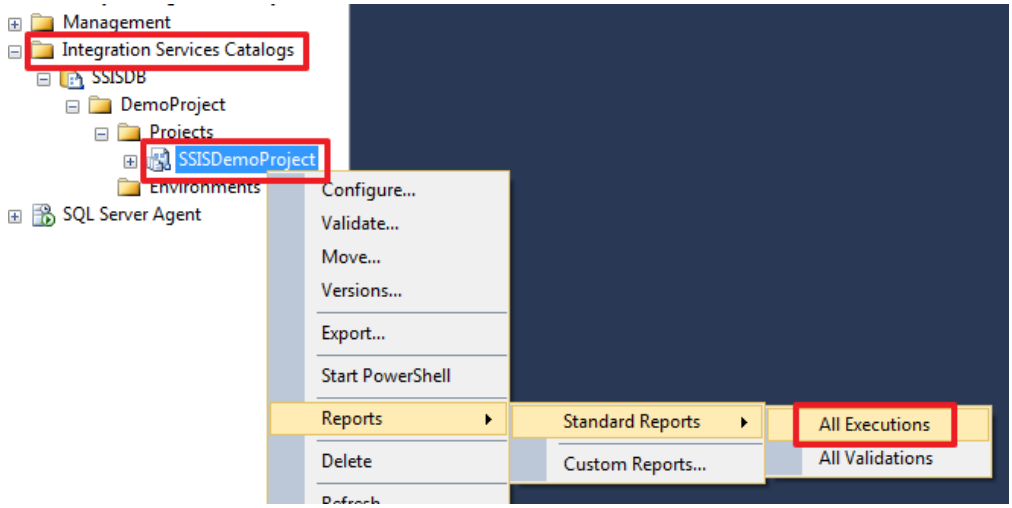
Так як крок у нас всього один, то задача запуститься відразу, інакше потрібно було б вказати з якого кроку потрібно почати виконання.

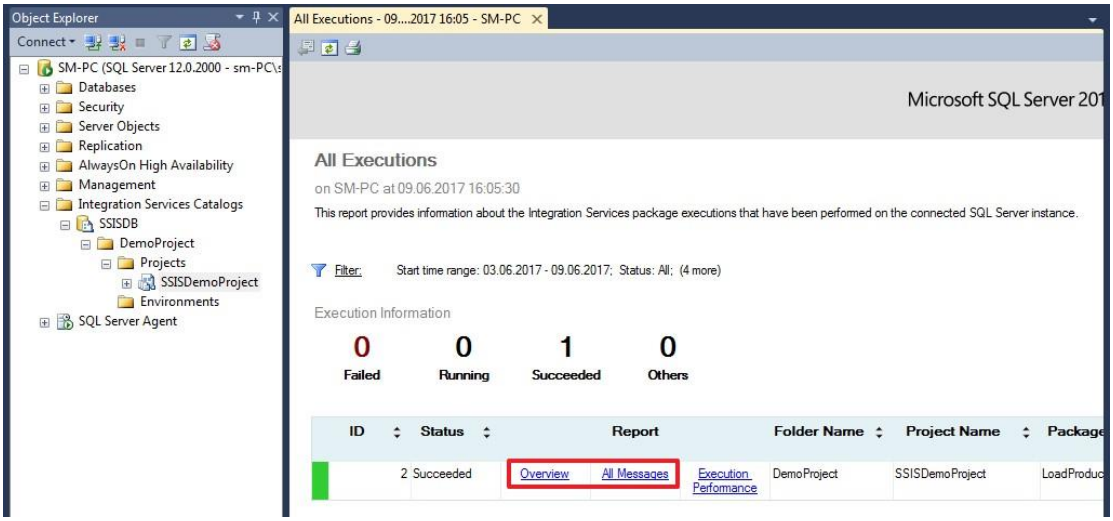
Результат виконання завдання можна побачити в наступному журналі:

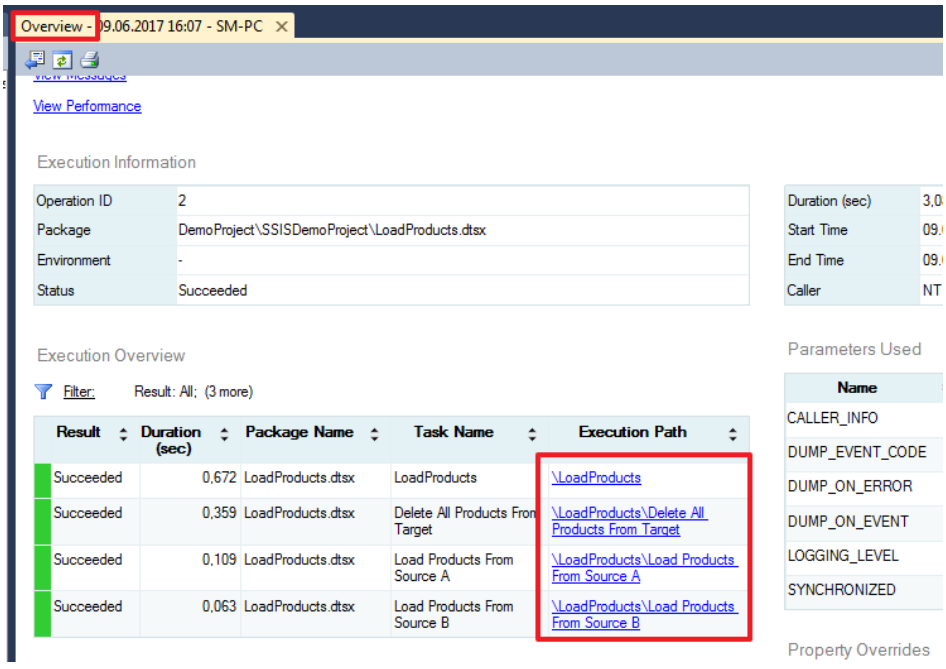


В даному журналі можна побачити успішність завершення кожного кроку, а також час виконання та інші параметри.

Докладніший звіт про виконання пакету можна подивитися за допомогою наступного звіту:







У цій частині змінимо логіку завантаження довідника Products:

За допомогою компонента «Union All» об'єднаємо два вхідних потоку в один; Для нових записів будемо робити вставку, а для записів, які вже були додані раніше будемо робити оновлення. Для поділу записів на додаються і оновлювані скористаємося компонентом Lookup;

Для поновлення записів застосуємо компонент «OLE DB Command».

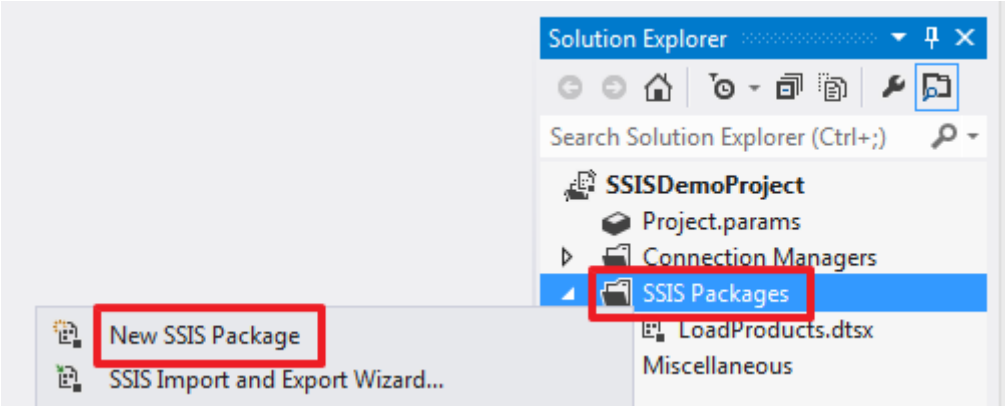
На завершення цієї частини розглянемо компонент Multicast для того щоб распараллелить виходить набір.

Разом в цій частині ми познайомимося з чотирма новими компонентами: Union All, Lookup, OLE DB Command і Multicast.

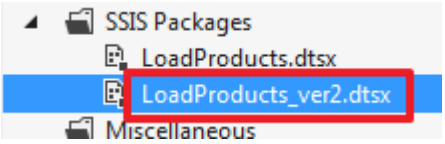
Далі так само буде дуже багато картинок.

Продовжуємо знайомство з SSIS

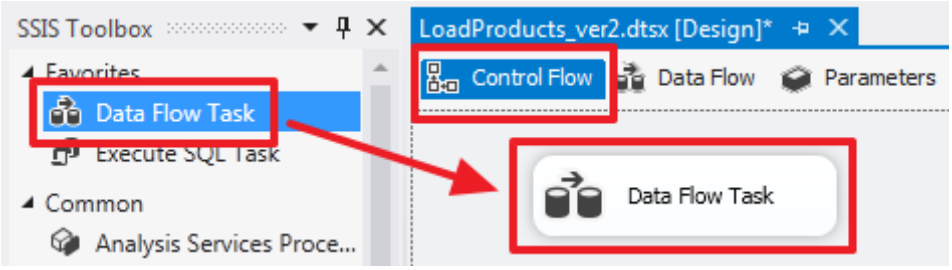
Створимо новий пакет:



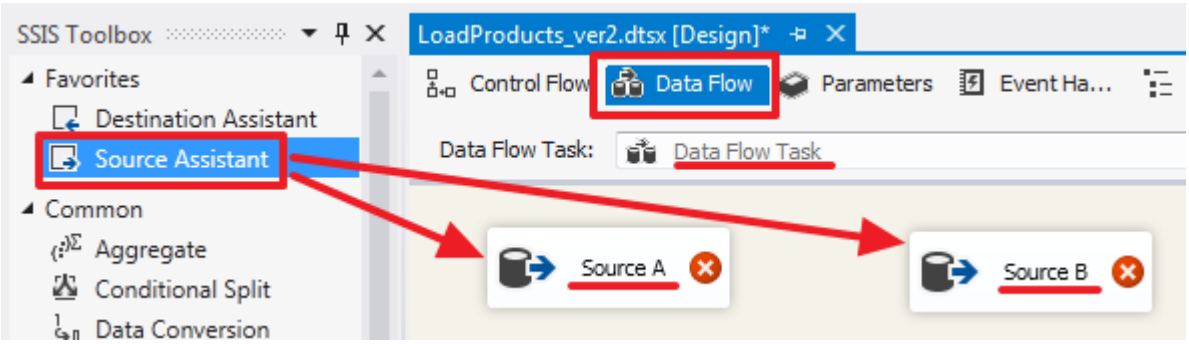
І перейменуємо його у «**LoadProducts_ver2.dtsx**»:



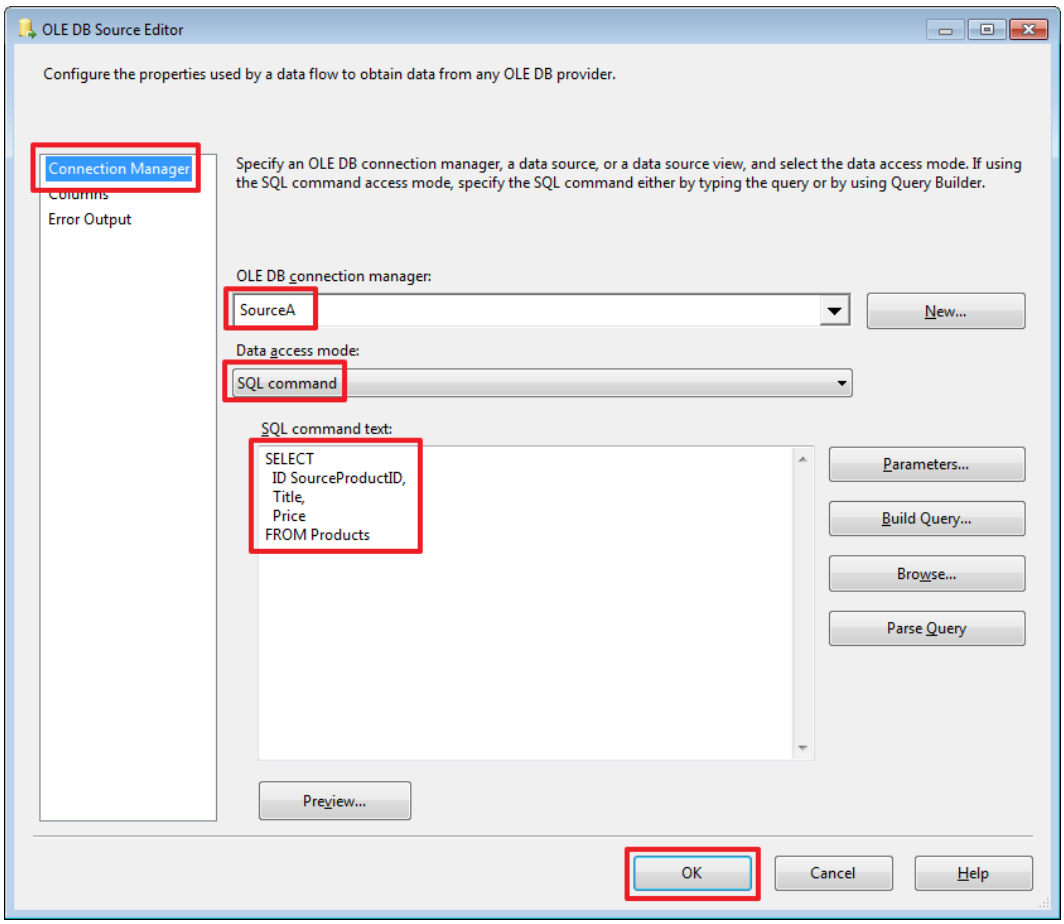
В області «**Control Flow**» створимо «**Data Flow Task**»:



Подвійним клацанням по елементу «Data Flow Task» зайдемо в його область «Data Flow». Створимо два елементи «Source Assistant» для з'єднань SourceA і SourceB. Перейменуємо ці елементи в «Source A» і «Source B» відповідно:



«Source A» налаштуємо наступним чином:

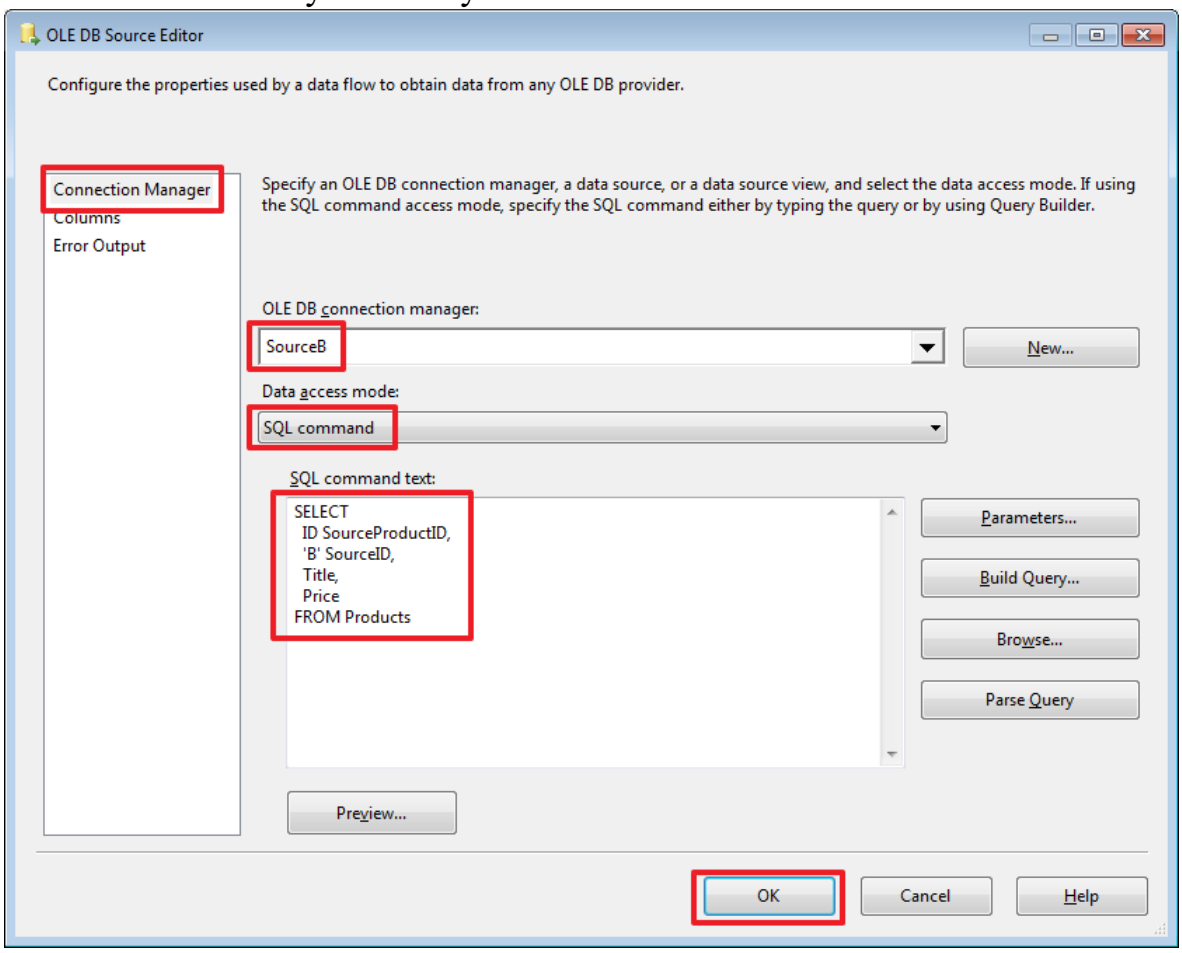


Текст запиту:

```
SELECT
ID SourceProductID,
  Title,
Price
FROMProducts
```

З метою демонстрації великих можливостей за раз, тут я навмисно відпустив SourceID.

«Source B» налаштуємо наступним чином:

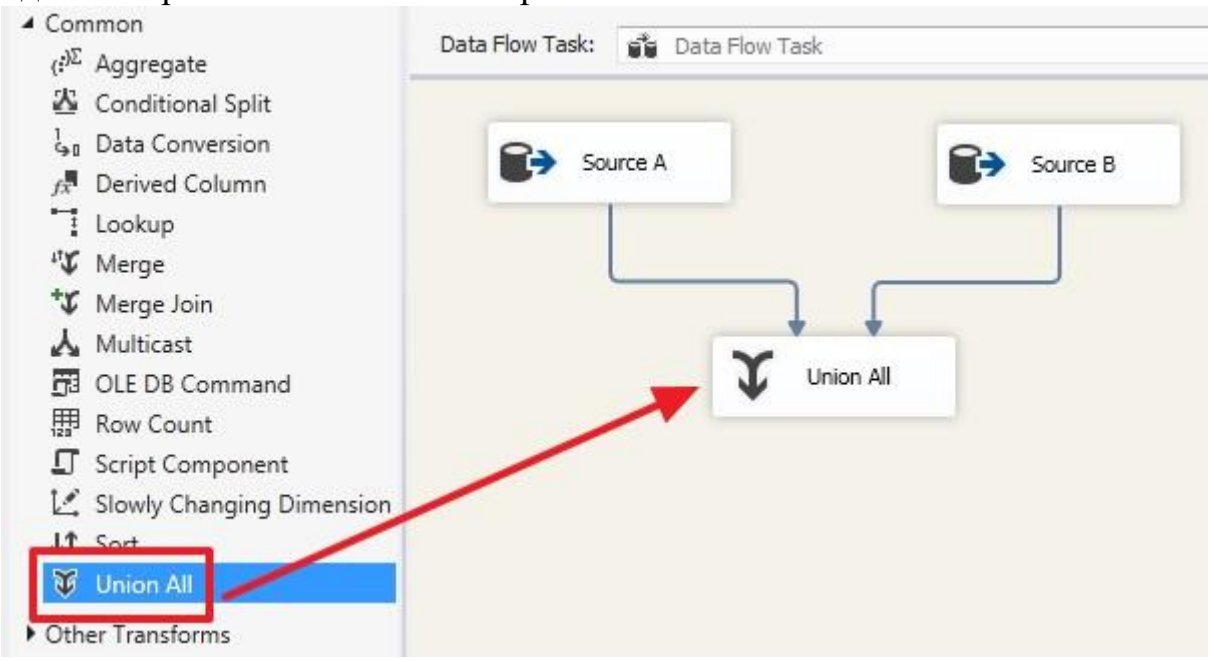


Текст запита:

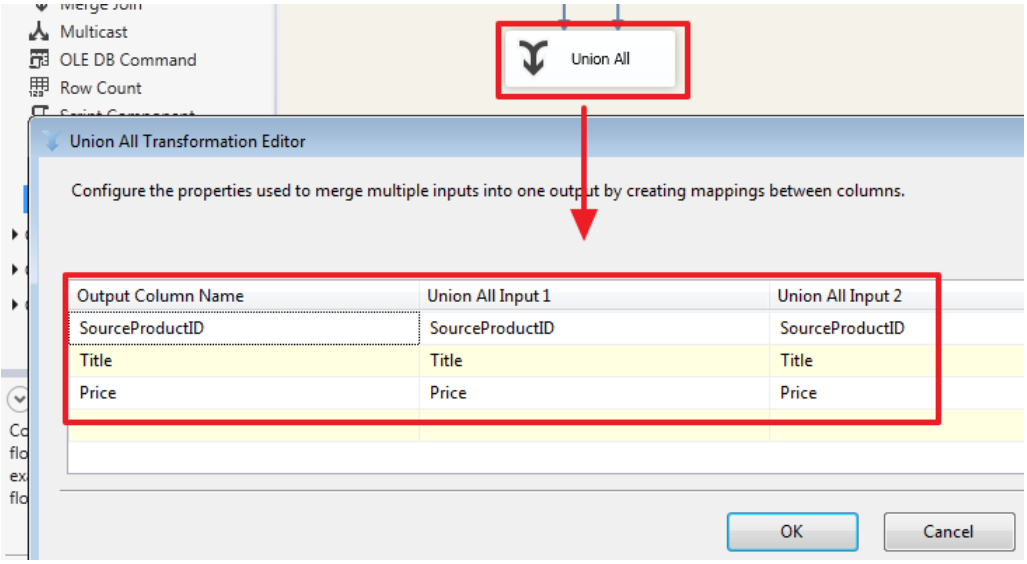
```
SELECT
ID SourceProductID,
'B' SourceID,
  Title,
  Price
FROM Products
```

В результаті набір A у нас буде мати 3 колонки [SourceProductID, Title, Price], а набір B матиме 4 колонки [SourceProductID, SourceID, Title, Price].

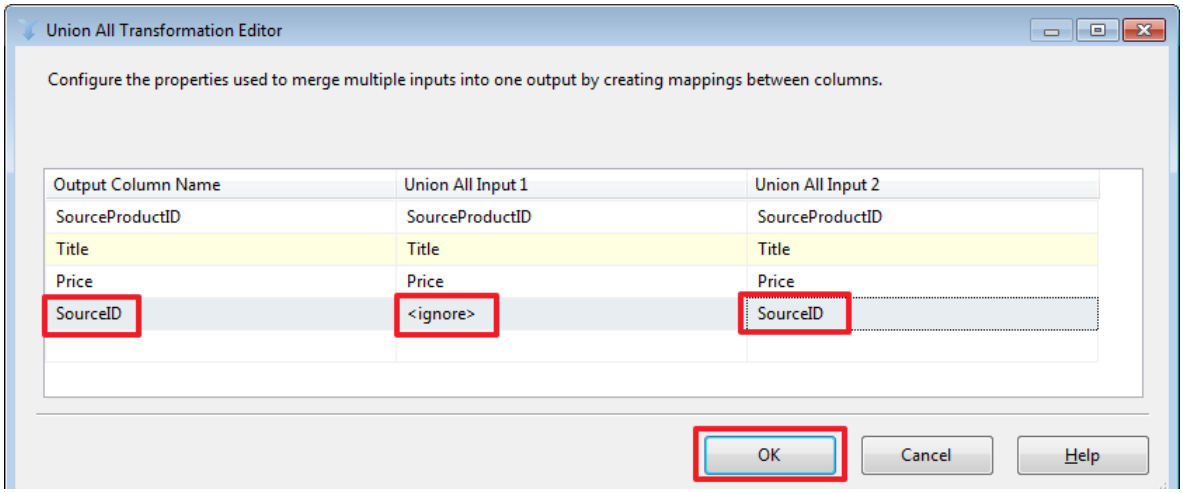
Скористаємося елементом «Union All», щоб об'єднати дані з 2-х наборів в один. Направимо в нього сині стрілки з «Source A» і «Source B»:



Яким чином робиться зіставлення колонок двох вхідних наборів, можна побачити двічі клацнувши на елементі «Union All»:



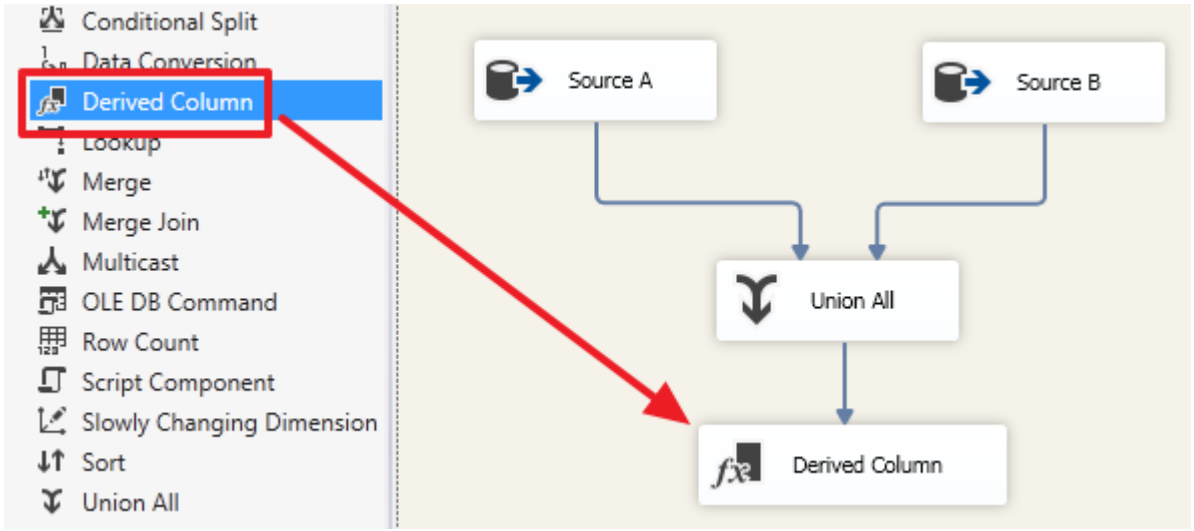
Як ми бачимо, тут зробилося автоматичне співставлення колонок імена яких збігаються. При необхідності ми можемо зробити своє зіставлення, для прикладу додамо колонку SourceID з другого набору:



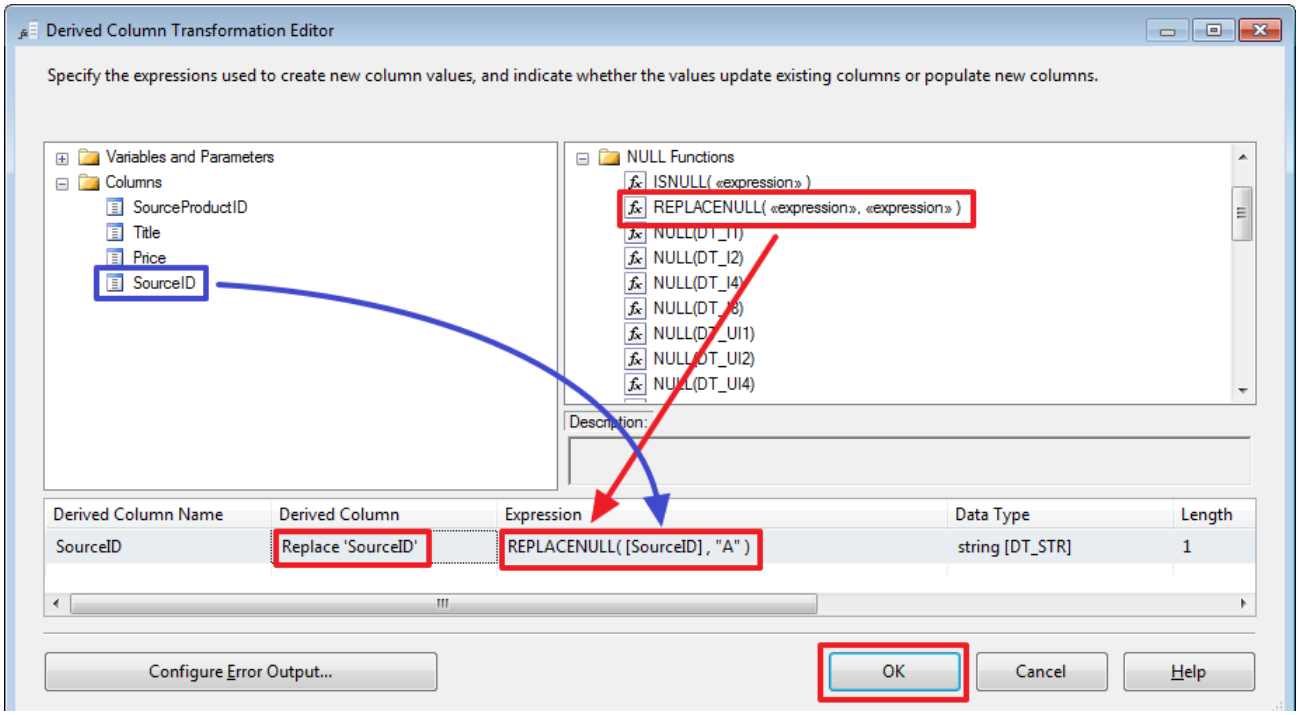
В даному випадку значення SourceID набору «Source A» дорівнюватимуть NULL.

Об'єднання двох наборів в даному випадку робиться на стороні SSIS. Тут варто звернути увагу на те, що бази джерела і приймаюча база можуть розташовуватися на різних серверах / примірниках SQL Server, з цієї причини ми не завжди зможемо так просто написати SQL запит використовуючи в ньому таблиці з різних баз із застосуванням SQL-операції UNION або JOIN (який можна було використовувати замість Lookup описаного нижче).

Для того щоб замінити NULL значення на «A» скористаємося компонентом «Derived Column» в який направимо потік з «Union All»:



Подвійним клацанням зайдемо в редактор «Derived Column» і налаштуємо його наступним чином:

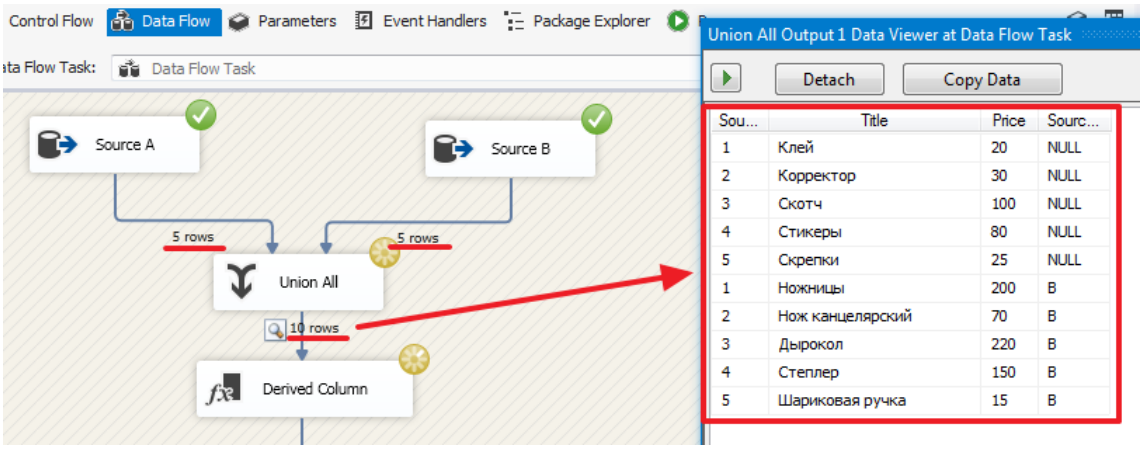


Зробимо наступне (миша в допомогу):

Вкажемо в «Derived Column» значення «Replace 'SourceID'» - це буде означати що ми на виході замінюємо стару колонку SourceID на нову;Перетягнемо в область «Expression» функцію REPLACENULL;Перетягнемо на місце першого аргументу функції REPLACENULL колонку SourceID

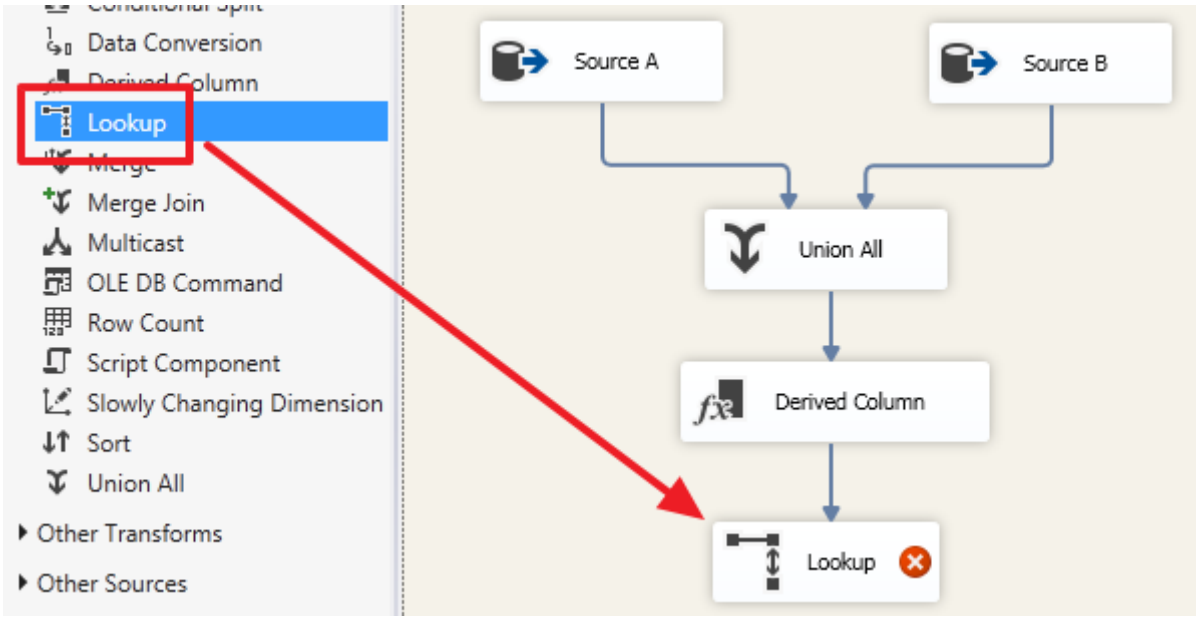
В якості другого аргументу пропишемо константу «A».Для того щоб зрозуміти, що сталося з даними після проходження «Union All» зробіть «Enable Data Viewer» для стрілки, що йде від «Union All» до «Derived Column»:

Тепер при запуску пакета на виконання ви зможете побачити набір, який вийшов в результаті:

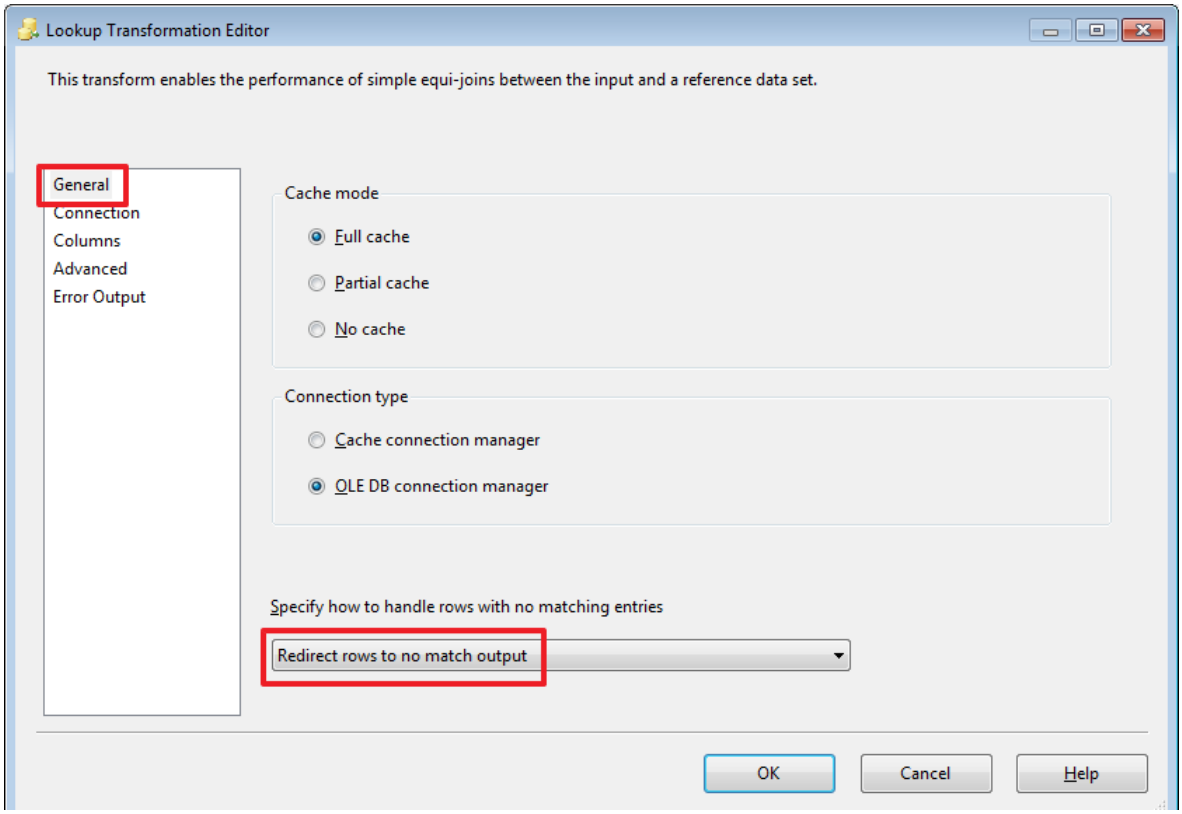


Тут видно, що на цьому етапі (до Derived Column) в колонці SourceID для рядків першого набору стоять значення NULL.

Для того щоб визначити чи була додана раніше запис в базу DemoSSIS_Target скористаємося компонентом Lookup:



Двічі клацнувши по ньому налаштуємо даний елемент:



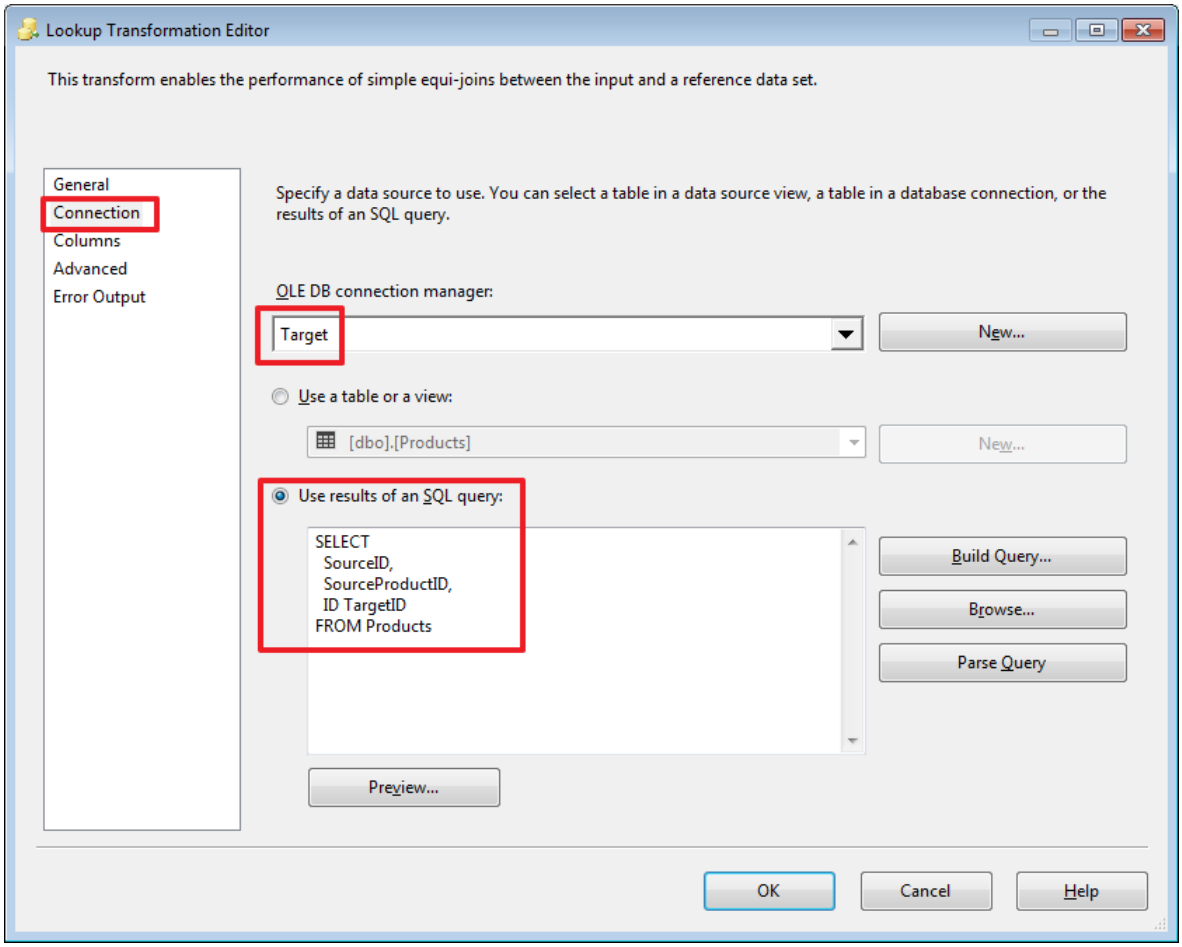
Тут ми скажемо, що ті рядки, для яких не знайдено відповідність, ми будемо перенаправляти в потік «no match output». У цьому випадку на виході ми отримаємо 2 набору «Lookup Match Output» і «Lookup No Match Output».

Наприклад, якщо виставити значення «Ignore failure», то в рядках, для яких не знайшлося зіставлення в поле TargetID (див. Нижче) буде записано значення NULL і всі рядки будуть повернуті через один набір «Lookup Match Output».

«Full cache» говорить про те, що набір, який буде використовуватися в якості довідника одним SQL запитом (см.на наступній вкладці) буде повністю завантажений в пам'ять і рядки будуть зіставлятися вже з кешу без повторних звернень до SQL Server.

Якщо ж вибрати «Partial cache» або «No cache», то на вкладці Advanced можна буде прописати запит з параметрами, який буде виконуватися для зіставлення кожного рядка вхідного набору. Для інтересу можна погратися з цією властивістю і через SQL Server Profiler подивитися які будуть формуватися запити при виконанні пакета.

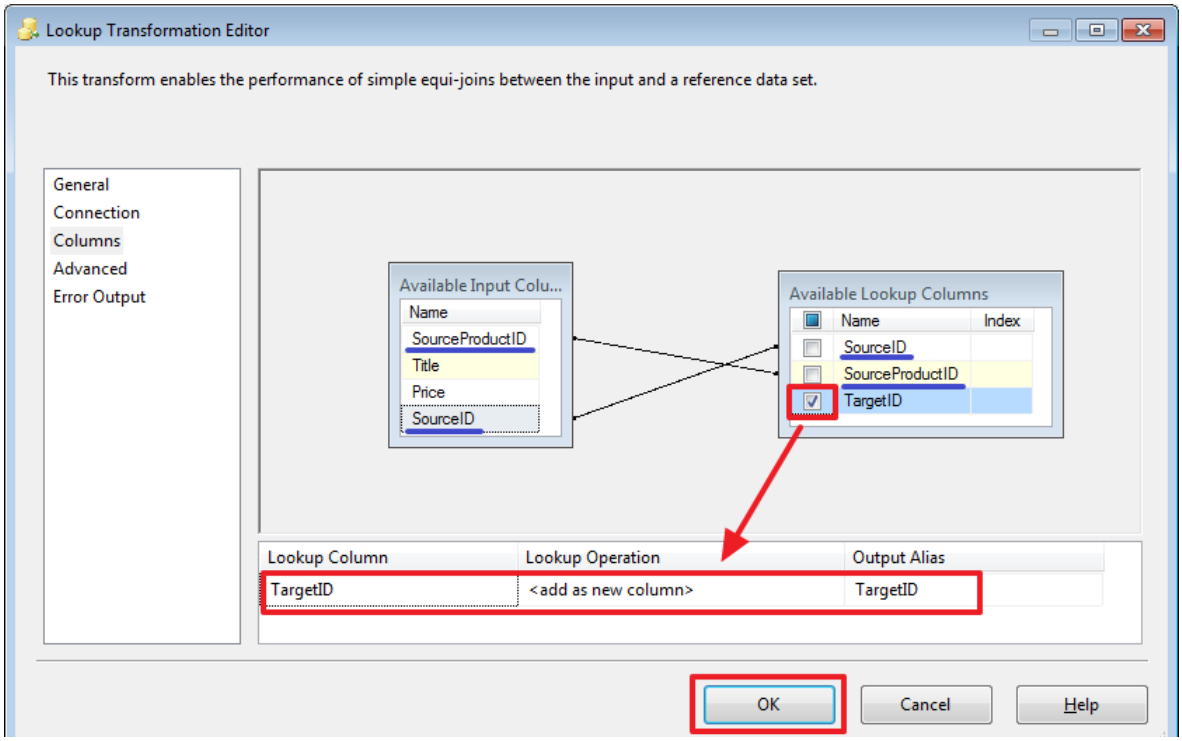
На наступній вкладці нам потрібно визначити набір, який буде виступати в ролі довідника:



Я прописав тут запит:

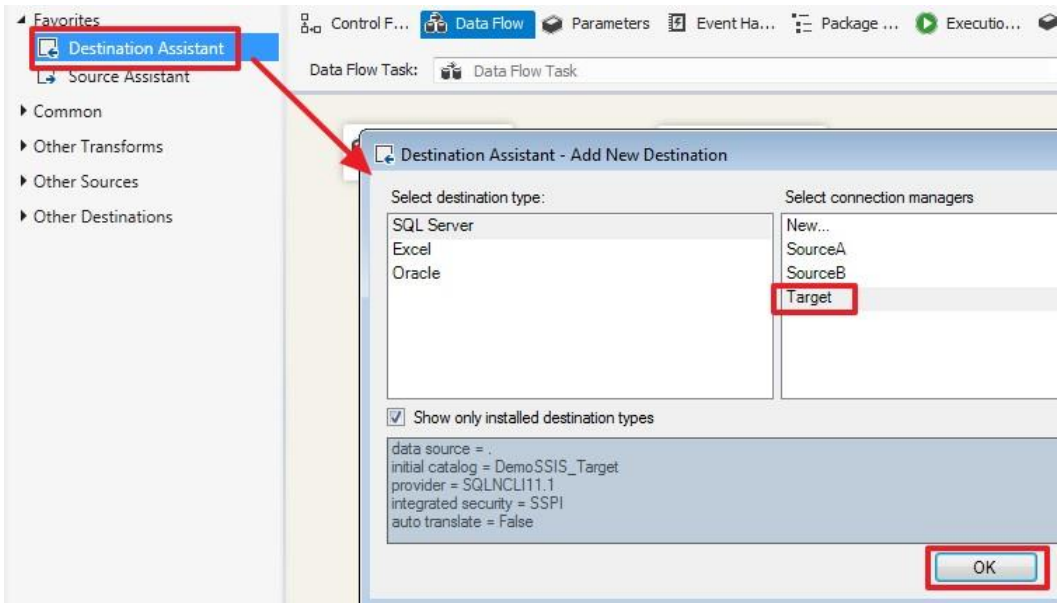
```
SELECT
SourceID,
SourceProductID,
IDTargetID
FROMProducts
```

На наступній вкладці потрібно вказати по яких полях робиться пошук рядка в довіднику і які колонки з довідника потрібно додати в вихідний набір (якщо це потрібно):

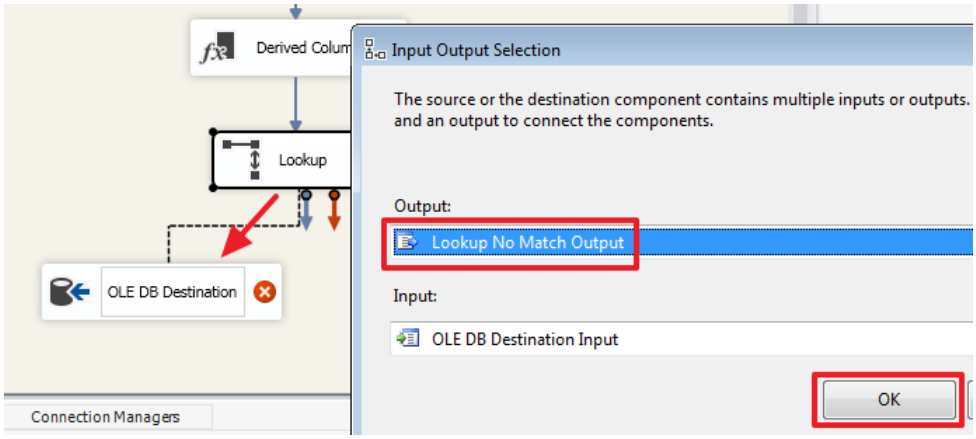


Для визначення зв'язку потрібно за допомогою миші перетягнути поле SourceProductID на SourceProductID і поле SourceID на SourceID.

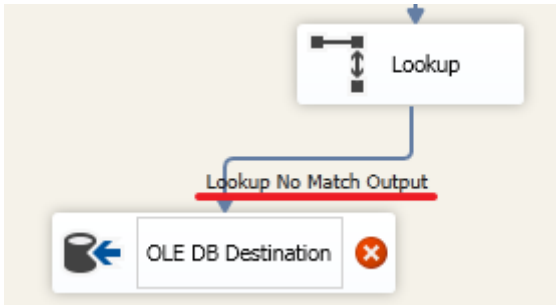
Додамо компонент «Destination Assistant» для вставки записів з потоку «Lookup No Match Output»:



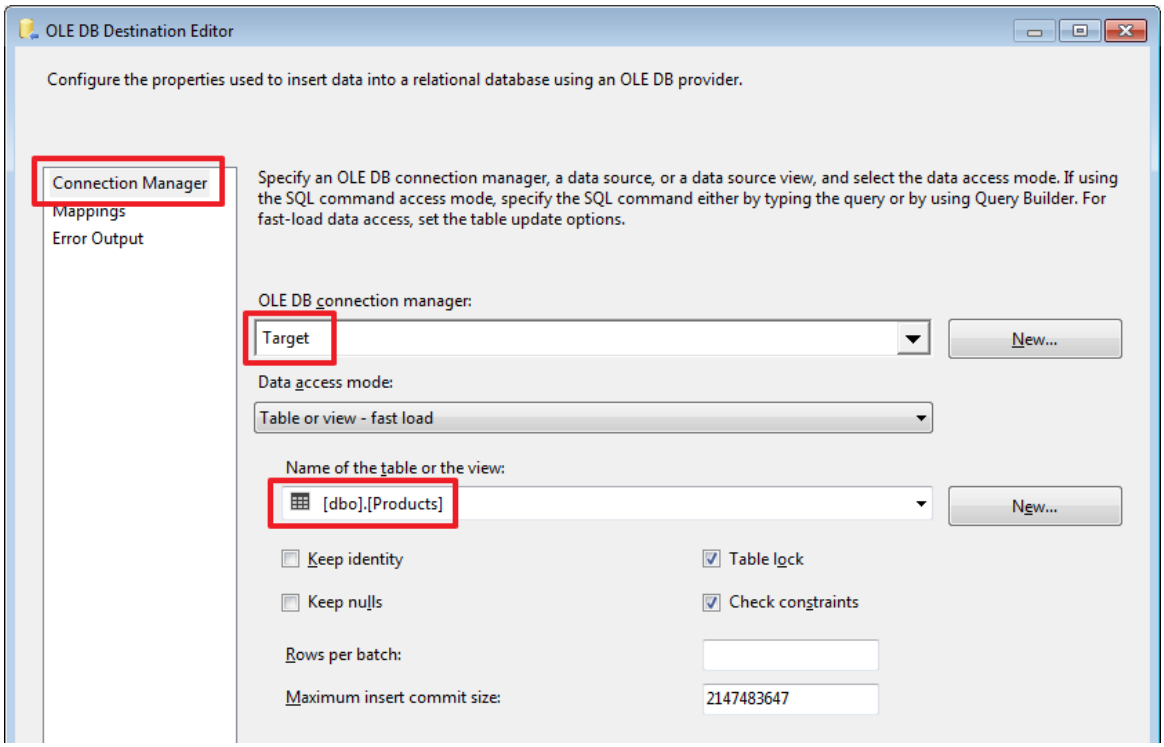
Перетягнемо синю стрілку з «Lookup» на «OLE DB Destination» і в діалоговому вікні виберемо потік «Lookup No Match Output»:

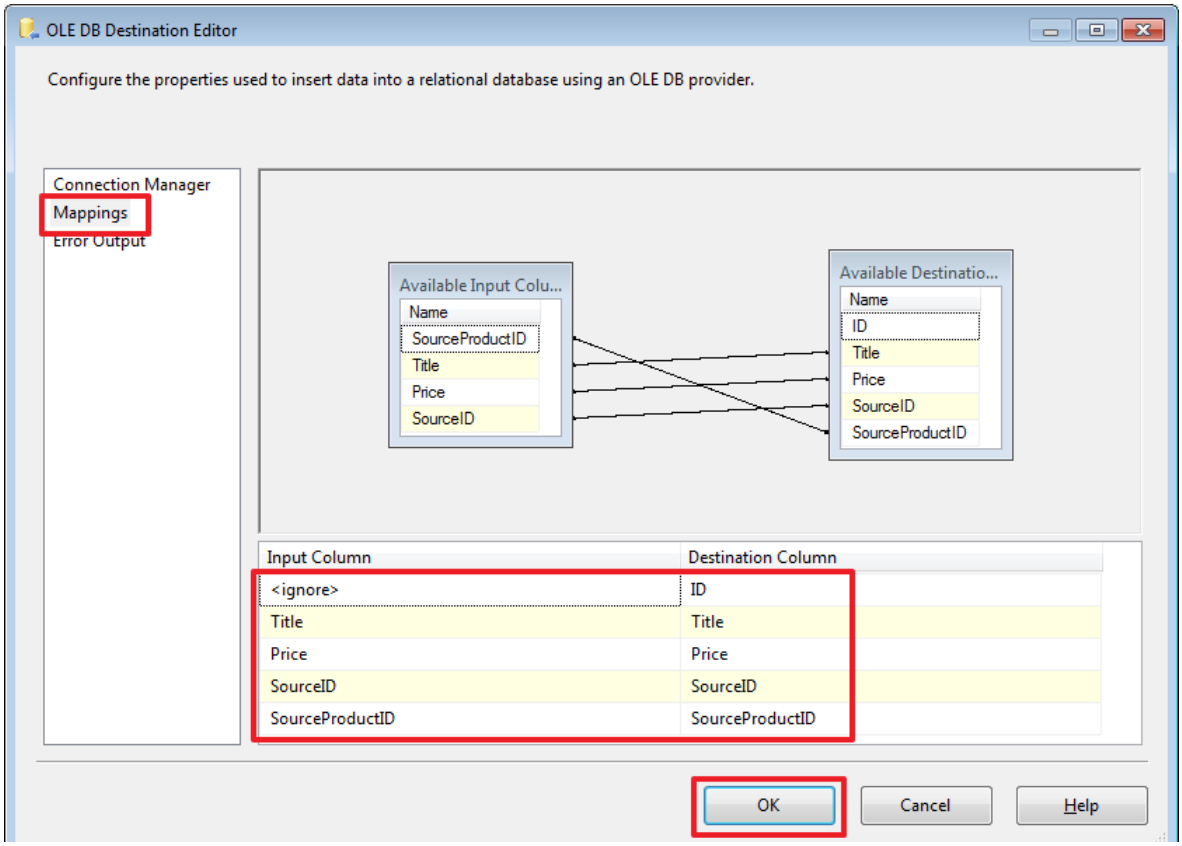


У підсумку ми отримаємо наступне:



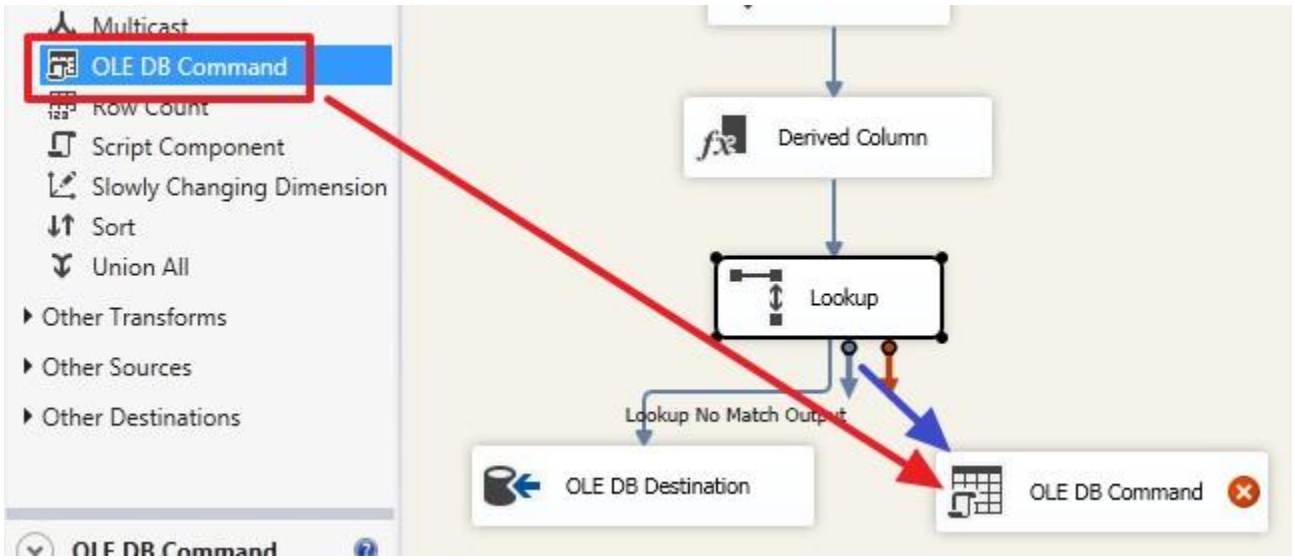
Двічі клацнувши по «OLE DB Destination» налаштуємо його:



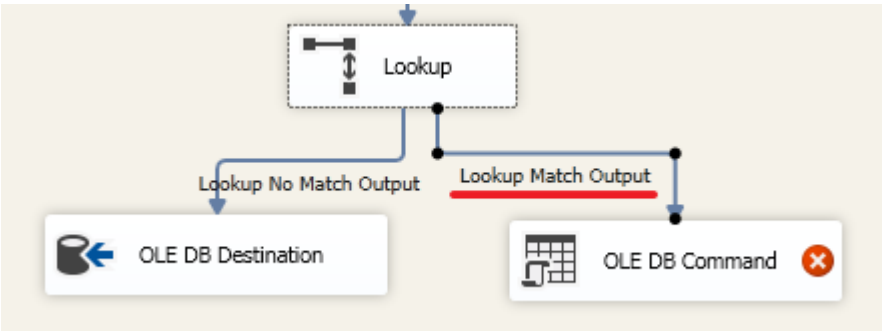


Обробку вставки нових записів ми зробили.

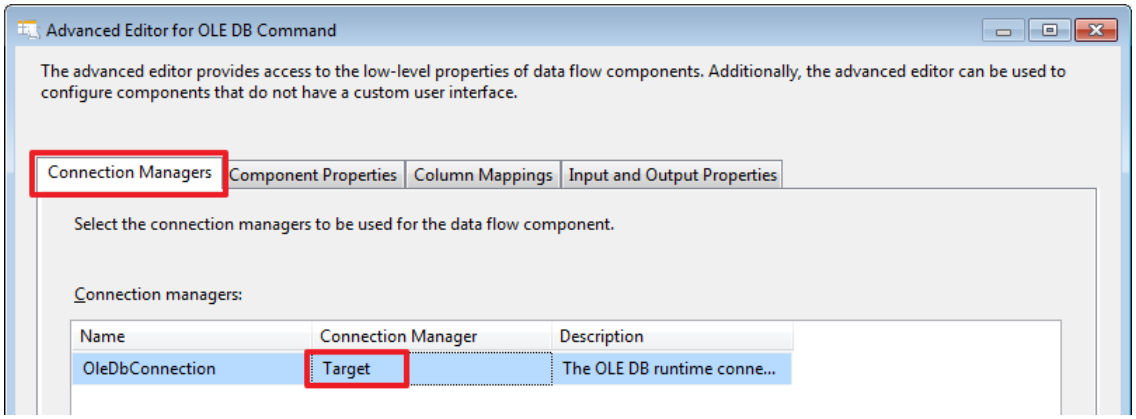
Тепер для поновлення раніше вставлених записів скористаємося компонентом «OLE DB Command» і перенесемо на нього синю стрілку від Lookup:

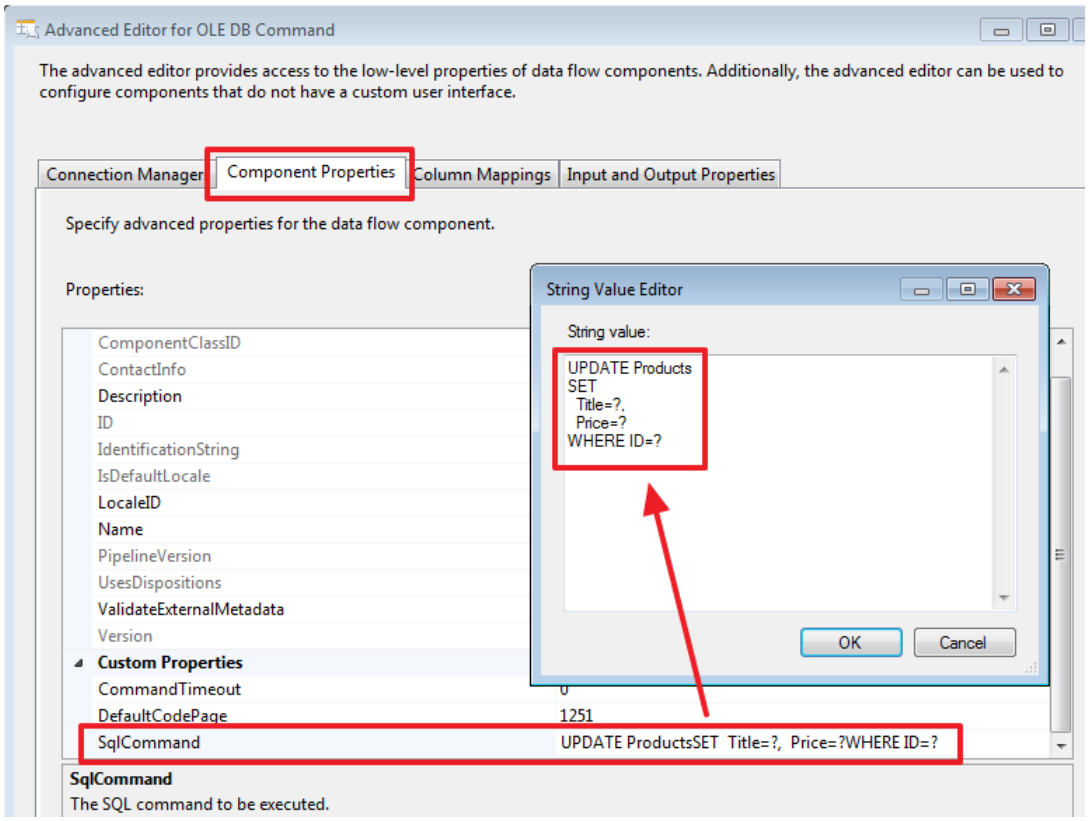


В цей компонент автоматично буде направлений потік «Lookup Match Output», тому що потік «Lookup No Match Output» ми вже вибрали раніше:



Двічі клацнемо на «OLE DB Command» і налаштуємо його:



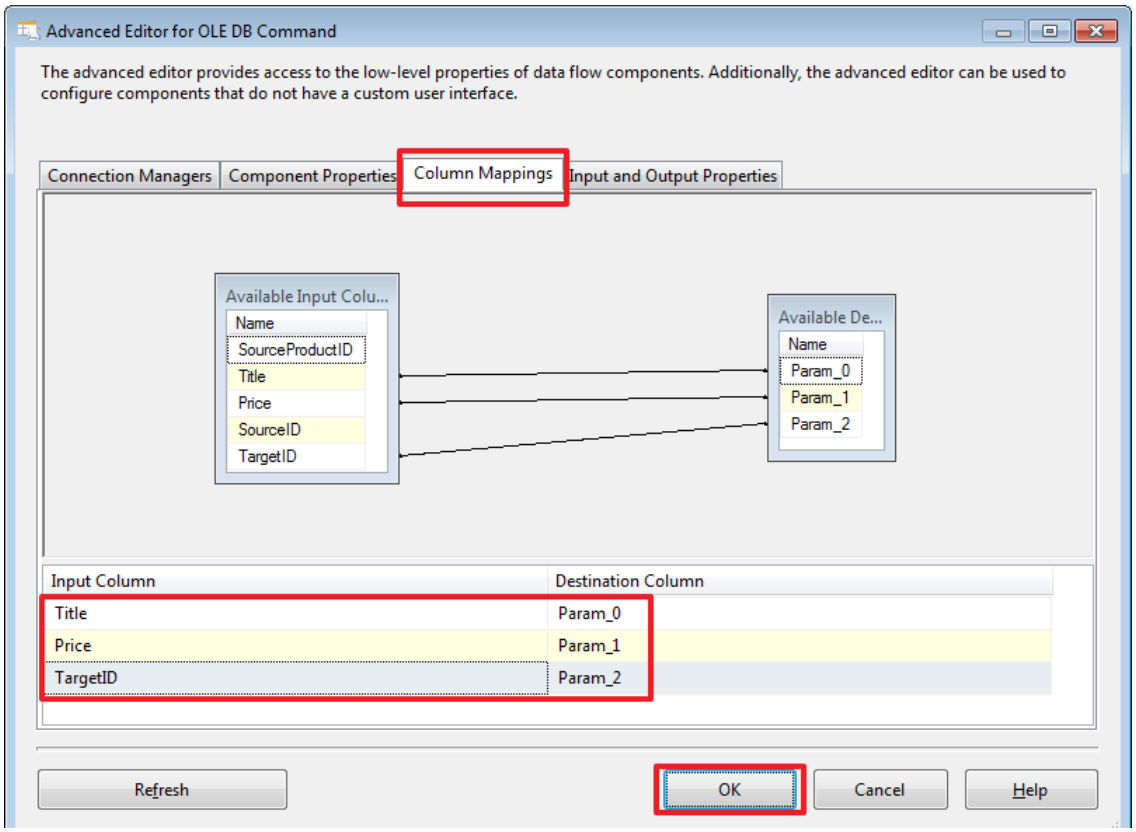


Пропишемо наступний запит на оновлення:

```
UPDATE Products
SET
```

```
 Title=?,
 Price=?
WHEREID=?
```

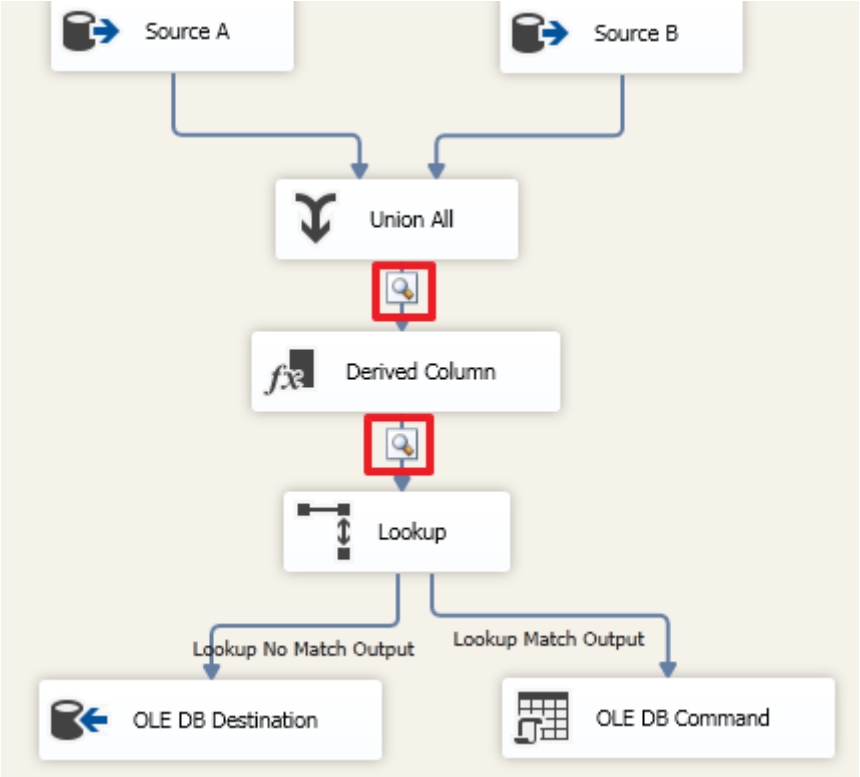
На наступній вкладці вкажемо яким чином будуть задаватися параметри на підставі даних рядків вхідного набору «Lookup Match Output»:



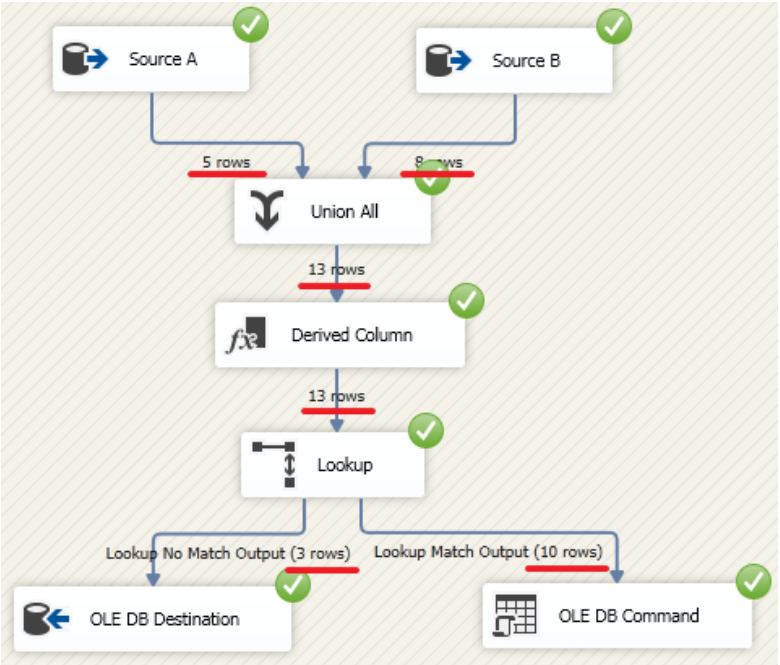
Через SSMS додамо нових продуктів в базу DemoSSIS_SourceB:

```
USE DemoSSIS_SourceB
GO

-- додамо нових товарів
SET IDENTITY_INSERT Products ON

INSERT Products(ID,Title,Price)VALUES
(6,N'Точилка',NULL),
(7,N'Ластик',NULL),
(8,N'Карандашпростой',NULL)

SET IDENTITY_INSERT Products OFF
GO
```

Для того щоб відстежити як змінювалися дані, ви можете, перед запуском пакета на виконання, в необхідних місцях зробити «Enable Data Viewer»:



Запустимо пакет на виконання:



У підсумку ми повинні побачити, що 3 рядки було вставлено за допомогою компонента «OLE DB Destination» і 10 рядків оновлено за допомогою компонента «OLE DB Command».

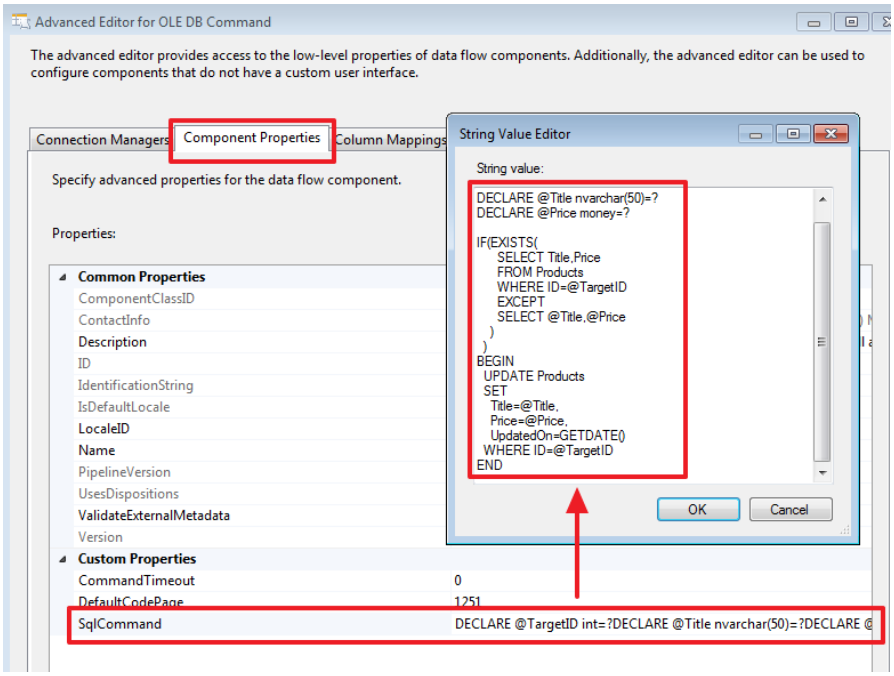
Запит прописаний в «OLE DB Command» виконався для кожного рядка вхідного набору, тобто в даному прикладі 10 разів.

У «OLE DB Command» можна прописати більш складну логіку на TSQL, наприклад, зробити перевірку, чи були змінені Title або Price, і робити оновлення відповідного рядка тільки якщо якесь із значень відрізняється.

Для наочності додамо нову колонку в таблицю Products в базі DemoSSIS_Target:

```
USE DemoSSIS_Target
GO

ALTERTABLE Products ADD UpdatedOn datetime
GO
```

Давайте тепер пропишемо наступну команду:



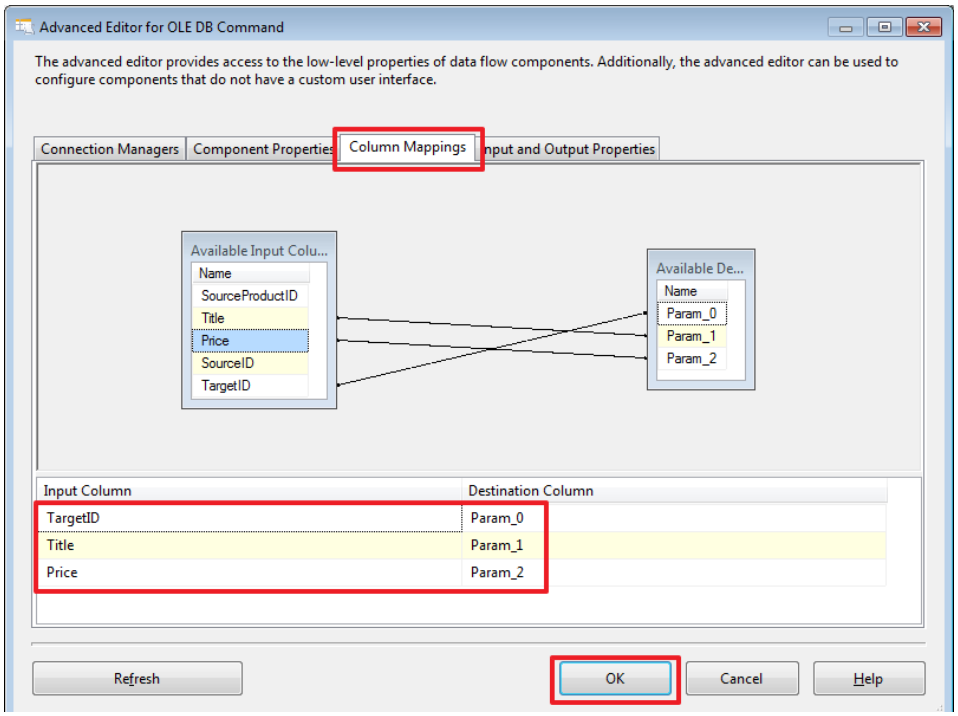
Текст команди:

```
DECLARE @TargetID int=?
DECLARE @Title nvarchar(50)=?
DECLARE @Price money=?

IF(EXISTS(
SELECTTitle,Price
FROM Products
WHEREID=@TargetID
EXCEPT
```

```
SELECT @Title,@Price
    )
  )
BEGIN
UPDATE Products
SET
    Title=@Title,
    Price=@Price,
    UpdatedOn=GETDATE()
WHEREID=@TargetID
END
```

Так само можна було б все це оформити у вигляді процедури, що, а тут прописати її через виклик «EXEC ProcName?,?,?». Тут, думаю, кому як зручніше, мені часом зручніше, щоб все було прописано в одному місці, тобто в SSIS-проект. Але якщо використовувати процедуру, то теж отримуємо свої зручності, в цьому випадку можна, було б просто змінити процедуру і уникнути переробки та повторного розгортання SSIS-проекту.

Після чого перевизначити прив'язку параметрів згідно їх черговості в тексті команди:

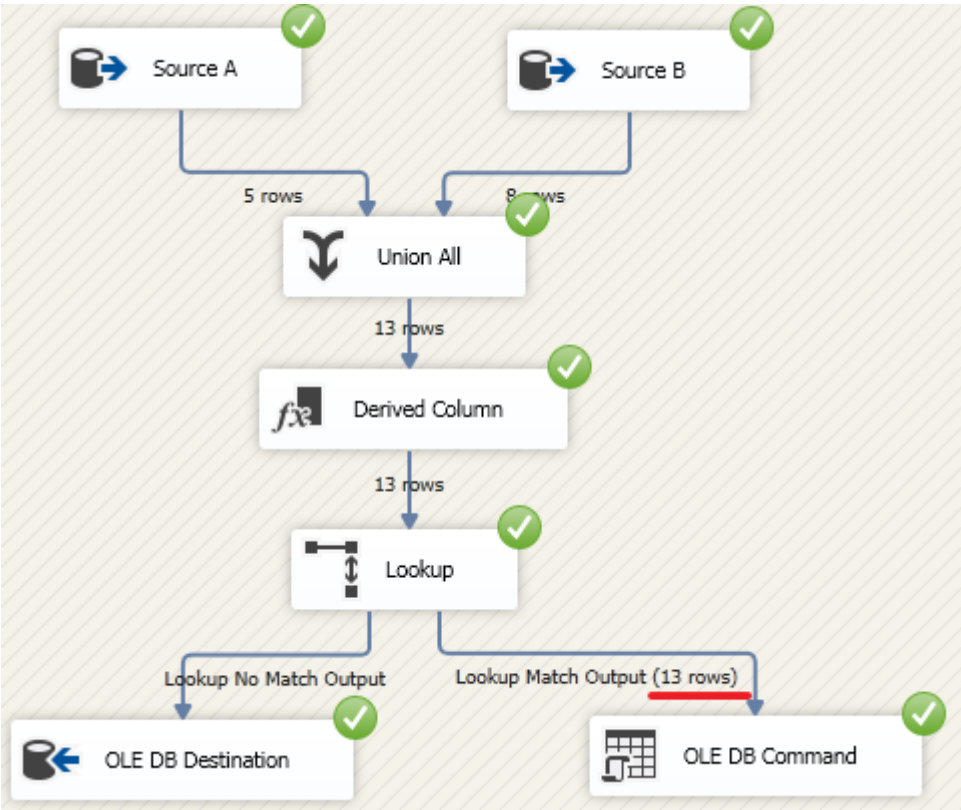


Зробимо в базі DemoSSIS_SourceA оновлення:

```
USE DemoSSIS_SourceA
```

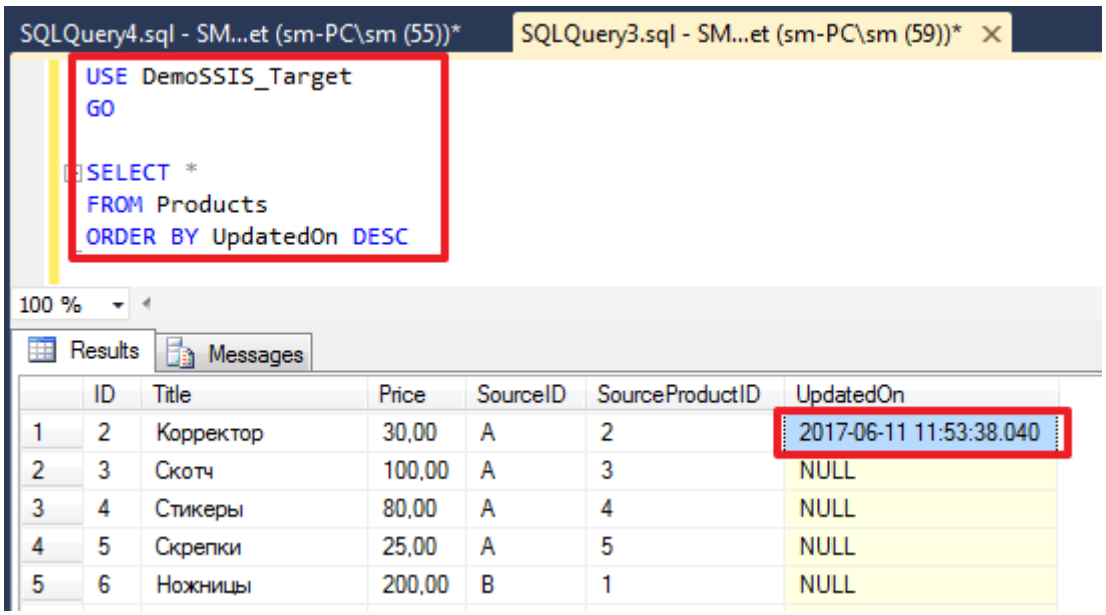
```
GO

UPDATE Products
SET
  Price=30
WHEREID=2-- Коректор
```

І знову запустимо проект на виконання. В результаті після чергового запуску пакета на виконання, UPDATE повинен буде виконатися тільки 1 раз, тільки для цього запису.



Після виконання пакету перевіримо це за допомогою запиту:

```
USE DemoSSIS_Target
GO

SELECT *
FROM Products
ORDERBY UpdatedOn DESC
```

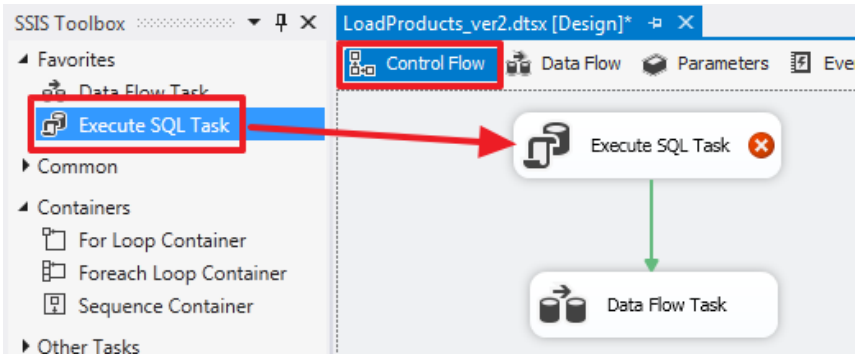


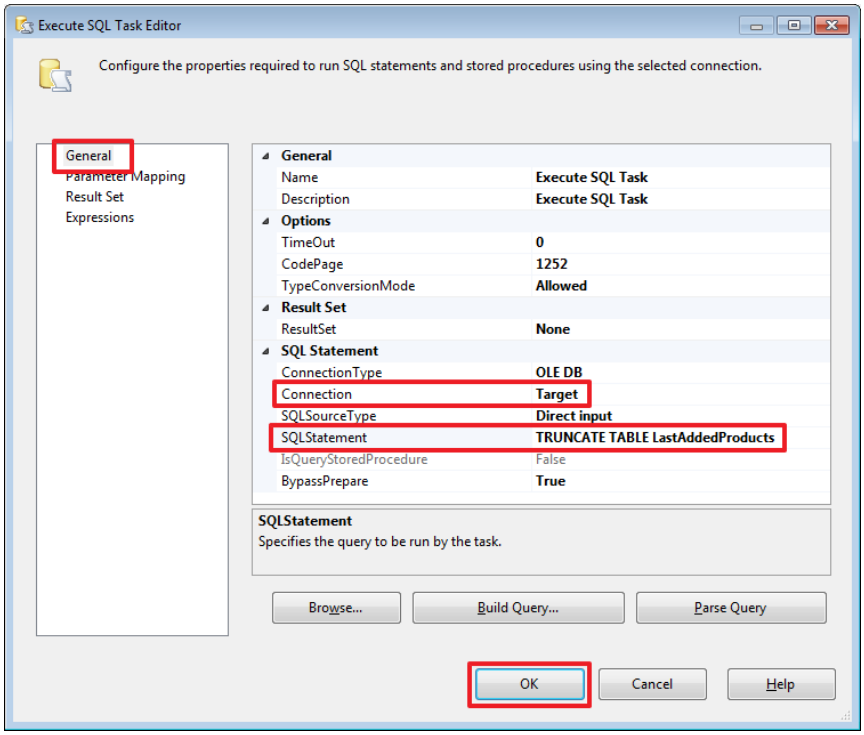
В рамках даної частини розглянемо ще компонент «Multicast». Даний компонент дозволяє отримати з одного потоку кілька. Це може бути корисно, коли одні і ті ж дані необхідно записати в два або більше різних місць - тобто входить один набір, а виходить стільки його копій скільки нам потрібно, і з кожною копією цього набору ми можемо робити що захочемо.

Для прикладу створимо в базі DemoSSIS_Target ще одну таблицю LastAddedProducts:

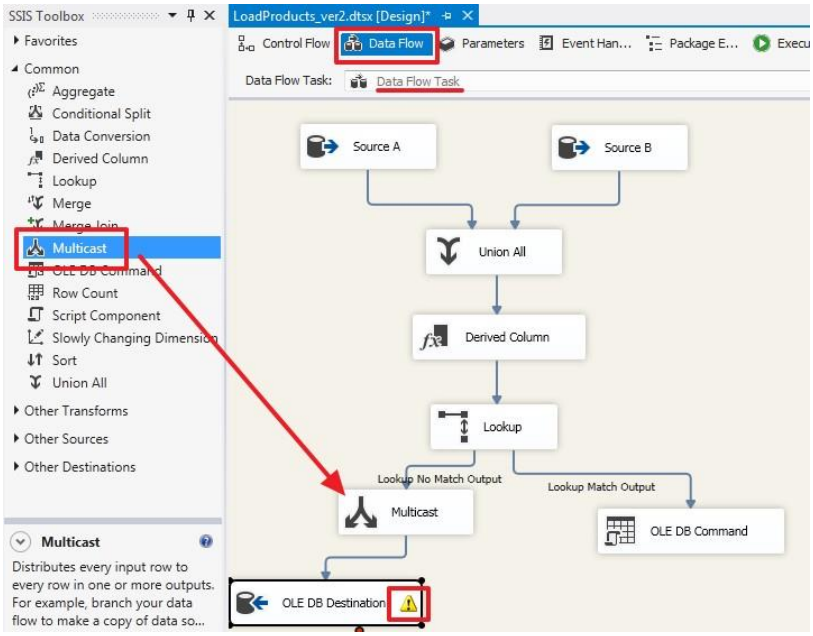
```
USE DemoSSIS_Target
GO

CREATETABLELastAddedProducts(
SourceID char(1) NOTNULL, -- використовується для ідентифікації джерела
SourceProductID intNOTNULL, -- ID в джерелі
  Title nvarchar(50) NOTNULL,
  Price money,
CONSTRAINT PK_LastAddedProducts PRIMARY
KEY(SourceID,SourceProductID),
CONSTRAINT CK_LastAddedProducts_SourceID CHECK(SourceID IN('A','B'))
)
GO
```

Для очищення цієї таблиці додамо в область «Control Flow» компонент «Execute SQL Task» і пропишемо в ньому команду «TRUNCATE TABLE LastAddedProducts»:

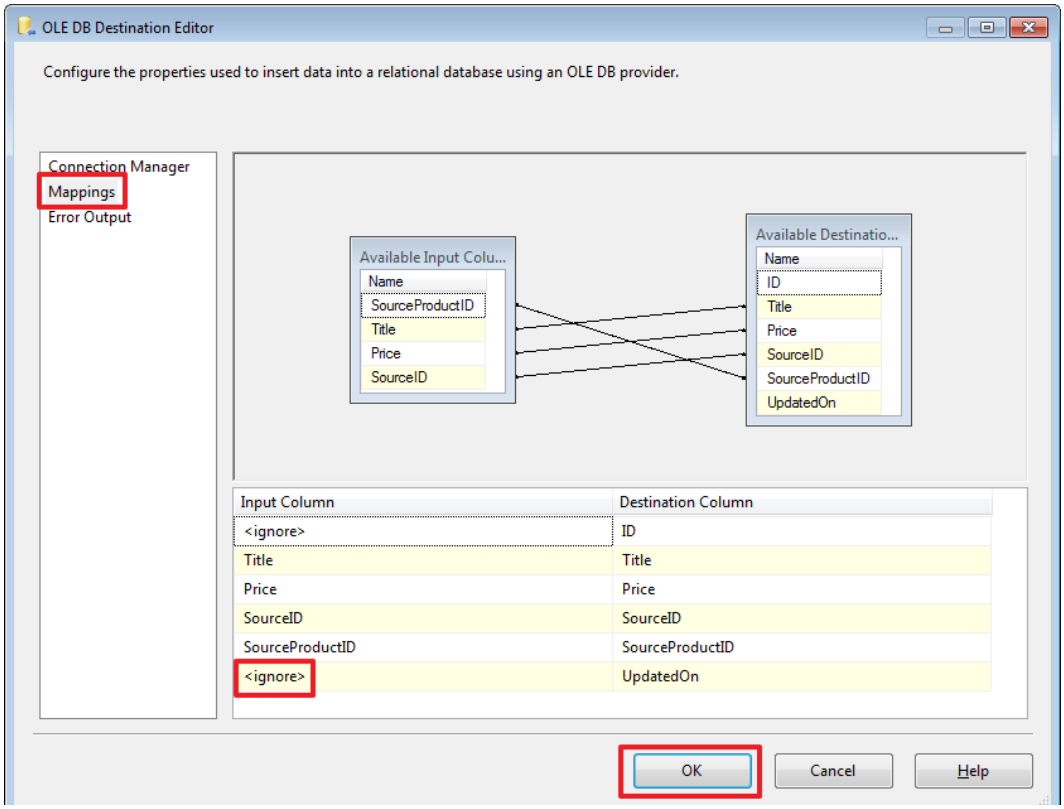




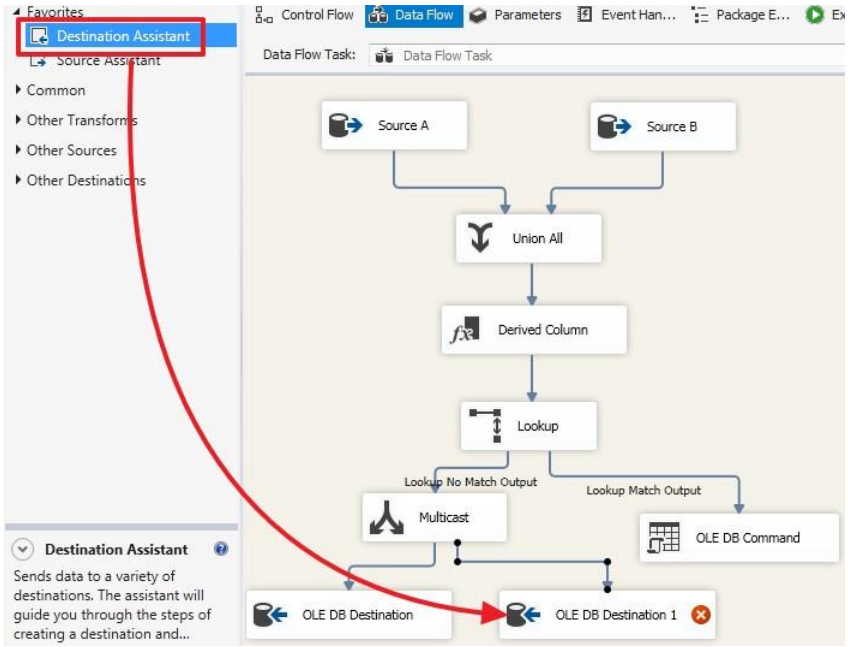
Перейдемо в область «Data Flow» компонента «Data Flow Task» і додамо компонент наступним чином:



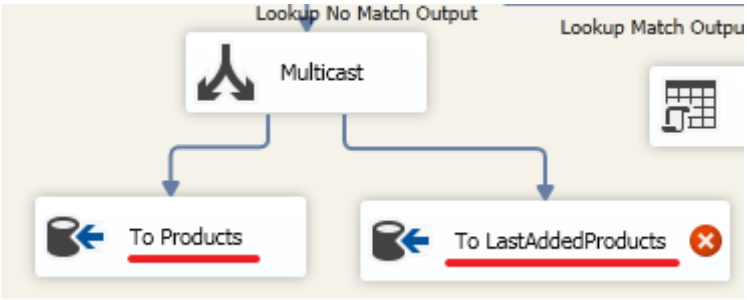
Зверніть увагу на жовтий знак оклику - це сталося через те, що ми додали колонку UpdatedOn і не прив'язали її. Зайдемо в елемент «OLE DB Destination», перейдемо на вкладку Mappings залишимо для колонки UpdatedOn як вхідного поля Ignore і натиснемо OK



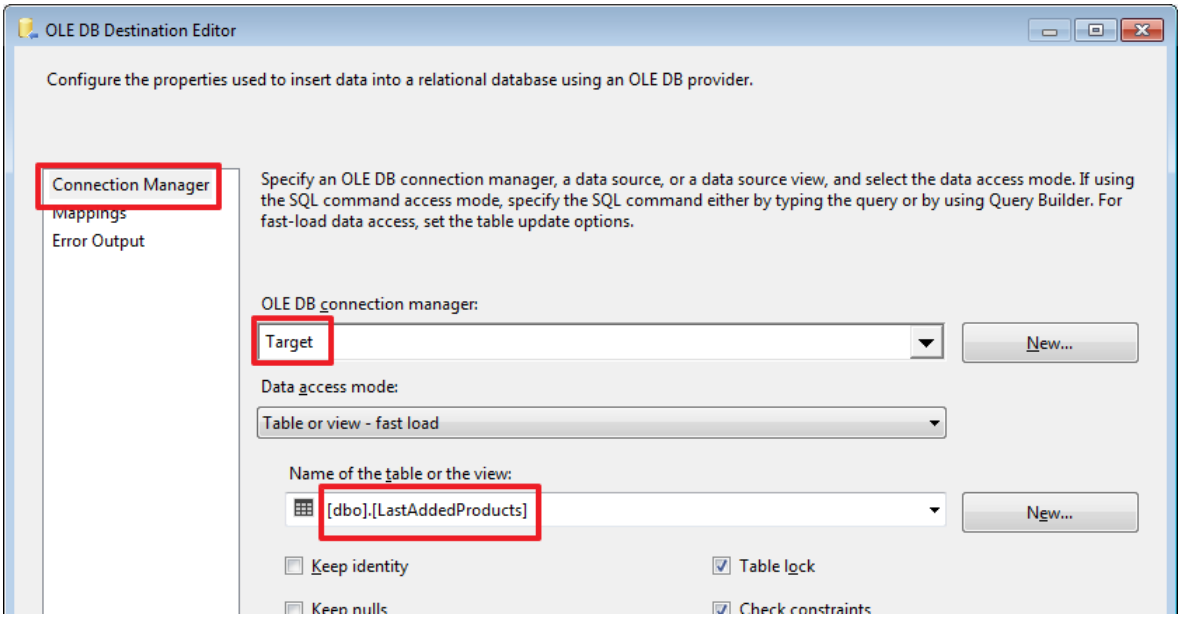
Створимо ще один елемент «OLE DB Destination» і перетягнемо на нього другу синю стрілку від елемента Multicast:

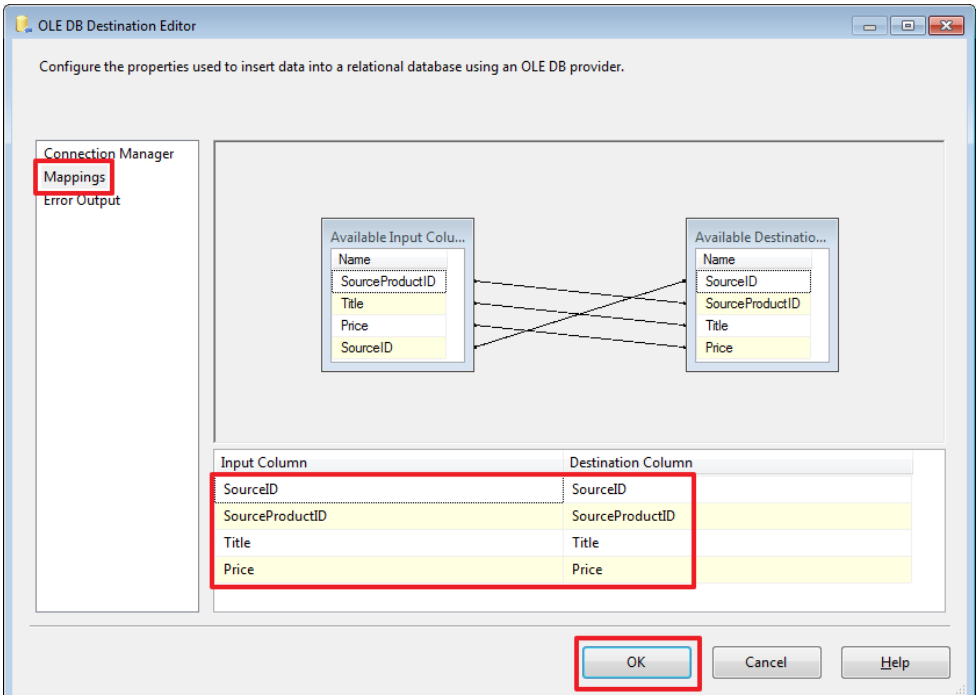


Перейменуємо для наочності:



Налаштуємо «To LastAddedProducts»:

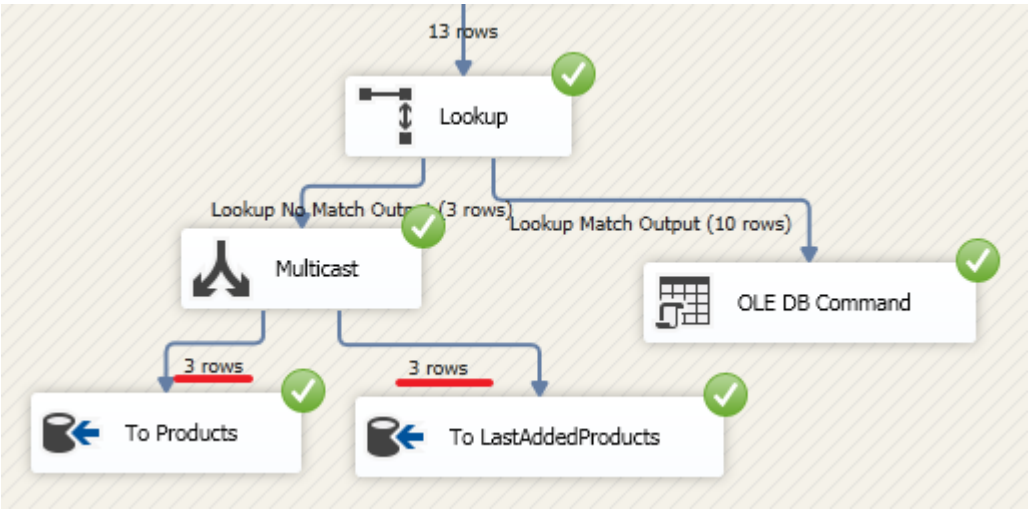




Видалимо через SSMS три останні вставлені записи:

```
USE DemoSSIS_Target
GO

DELETE Products
WHERE SourceID='B'
AND SourceProductID>=6
```

І запустимо пакет на виконання:



В результаті додавання відбулося в 2 таблиці - Products і LastAddedProducts.

SSIS досить цікавий інструмент, який на мій погляд не завадить мати всвоєму арсеналі, так як в деяких випадках він може сильно спростити процес інтеграції. Але звичайно бувають ситуації, коли все зваживши, розумніше написати інтеграцію вдаючись до інших способів, наприклад, використовувати Linked Servers і писати процедури на чистому TSQL або писати свою утиліту на якомусь іншому мовою програмування із застосуванням всієї потужності ООП і т.п.