



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



**.Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego
Projekt PO KL 4.1.2/2009 "Inżynier pilnie poszukiwany" nr umowy: UDA-POKL.04.01.02-00-141/09-00**

Zaawansowane programowanie baz danych

dr inż. Piotr Ratuszniak ratusz@ie.tu.koszalin.pl

KATEDRA INŻYNIERII KOMPUTEROWEJ

WYDZIAŁ ELEKTRONIKI I INFORMATYKI

POLITECHNIKA KOSZALIŃSKA

2009-2013

Celem prezentacji jest przedstawienie podstawowych zagadnień w ważnej tematyce programowania baz danych na przykładzie języka Transact SQL (T-SQL) w systemie zarządzania relacyjnymi bazami danych - Microsoft SQL Server 2008.

Specjalność programowania baz danych wraz ze specjalnością administracji baz danych stanowią jedne z podstawowych grup specjalności powiązanych z tematyką baz danych (dwie osobne ścieżki certyfikacji w Microsoft).

Uwaga: Firmy programistyczne zatrudniają również specjalistów na dedykowanych stanowiskach np.: „**Programista** baz danych”.

- Funkcje definiowane przez użytkownika
- Procedury składowane
- Wyzwalacze
- Transakcje, współbieżność i blokady
- Obsługa błędów
- Dane XML

Funkcje definiowane przez użytkownika

Własności

Strona 4

Funkcje UDF (*ang. User Defined Function*) mogą być umieszczone w:

- zapytaniach
- ograniczeniach
- kolumnach obliczeniowych

Właściwości UDF:

- funkcje UDF nie mogą modyfikować danych w tabelach
- funkcje UDF mogą tworzyć zmienne tabelaryczne
- mogą być tworzone z wykorzystaniem klauzuli EXECUTE AS (kontekst bezpieczeństwa dla wykonania funkcji)

Funkcje definiowane przez użytkownika

Własności

Strona 5

Funkcje UDF dzielimy na:

- Funkcje UDF T-SQL – utworzone w wykorzystaniu języka T-SQL, wspierają opcję :
 - ENCRYPTION – tekst funkcji zostaje przekonwertowany do nieczytelnego formatu
 - SCHEMABINDING - powiązanie z obiektami bazowymi co uniemożliwia usunięcie lub modyfikację schematu tych obiektów
- Funkcje UDF CLR - zdefiniowana z wykorzystaniem wybranego języka z platformy .NET
- Skalarne – zwracają pojedynczą skalarną wartość
- Tabelaryczne – zwracają tabelę, stosowane głównie w klauzuli FROM zewnętrznego zapytania SQL. Wbudowana tabelaryczna funkcja UDF przypomina widok, jednak może ona przyjmować parametry.
- Funkcje UDF wykonywane dla każdego wiersza

Funkcje definiowane przez użytkownika

Własności

Strona 6

Funkcje UDF T-SQL – koncertują się zazwyczaj na przetwarzaniu zbiorów danych

- Funkcje UDF – utworzone w wykorzystaniu języka T-SQL, wspierają opcję :
 - ENCRYPTION – tekst funkcji zostaje przekonwertowany do nieczytelnego formatu
 - SCHEMABINDING - powiązanie z obiektami bazowymi co uniemożliwia usunięcie lub modyfikację schematu tych obiektów
- Funkcje UDF CLR - zdefiniowana z wykorzystaniem wybranego języka z platformy .NET
- Skalarne – zwracają pojedynczą skalarną wartość
- Tabelaryczne – zwracają tabelę, stosowane głównie w klauzuli FROM zewnętrznego zapytania SQL. Wbudowana tabelaryczna funkcja UDF przypomina widok, jednak może ona przyjmować parametry.

Funkcje definiowane przez użytkownika

Przykład funkcji skalarnej

Strona 7

```
USE AdventureWorks;

go

CREATE FUNCTION dbo.employee_name(@employeeid AS INT)
RETURNS VARCHAR(100)
AS
BEGIN
DECLARE @employee AS VARCHAR(100)
SET @employee='';
SELECT @employee= Person.Contact.FirstName+Person.Contact.LastName FROM
Person.Contact,HumanResources.Employee
WHERE Person.Contact.ContactID=HumanResources.Employee.ContactID
AND HumanResources.Employee.Employeeid=@employeeid
RETURN @employee
END
```

Funkcje definiowane przez użytkownika

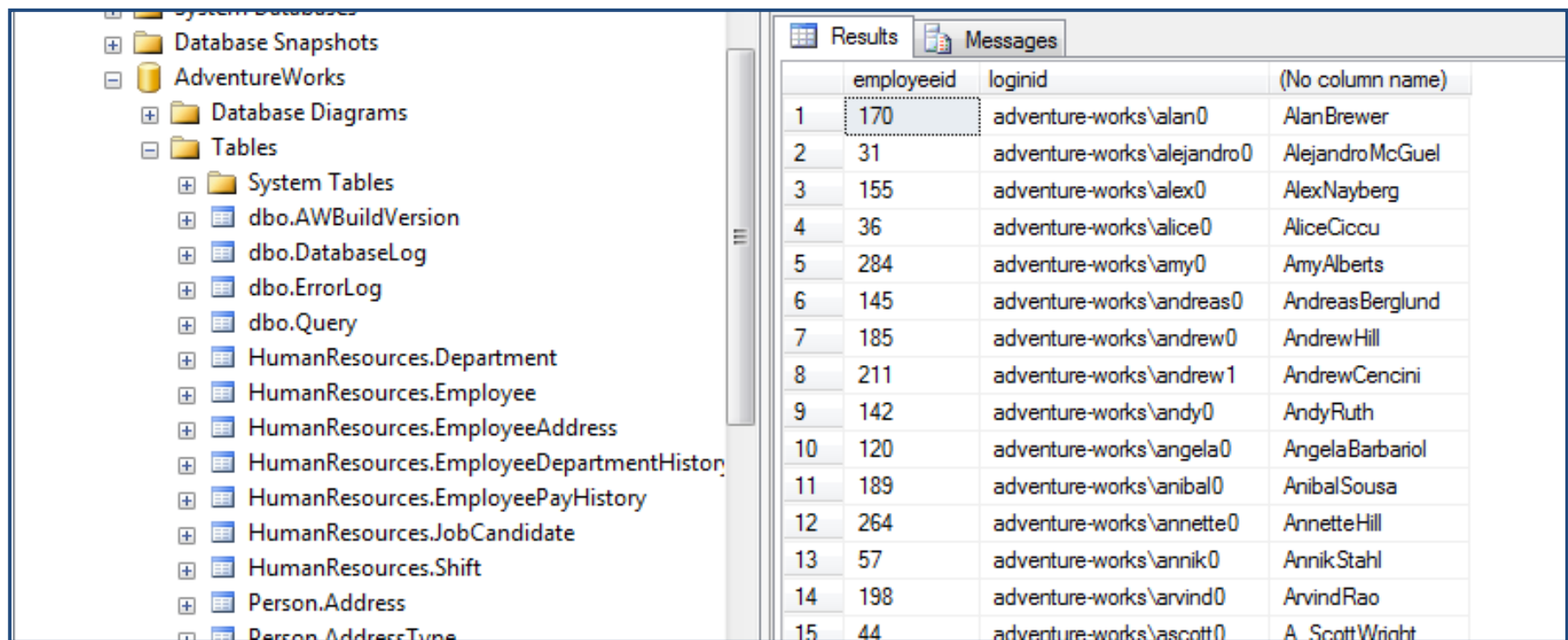
Przykład funkcji skalarnej - testowanie

Strona 8

```
USE AdventureWorks;
```

```
go
```

```
SELECT employeeid,loginid,dbo.employee_name(employeeid) from  
HumanResources.Employee;
```



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'AdventureWorks' database is expanded, showing its 'Tables' folder. The 'HumanResources.Employee' table is highlighted. On the right, the 'Results' window shows the output of the query. The table has three columns: 'employeeid', 'loginid', and '(No column name)'. The data is as follows:

	employeeid	loginid	(No column name)
1	170	adventure-works\alan0	AlanBrewer
2	31	adventure-works\alejandro0	AlejandroMcGuel
3	155	adventure-works\alex0	AlexNayberg
4	36	adventure-works\alice0	AliceCiccu
5	284	adventure-works\amy0	AmyAlberts
6	145	adventure-works\andreas0	AndreasBerglund
7	185	adventure-works\andrew0	AndrewHill
8	211	adventure-works\andrew1	AndrewCencini
9	142	adventure-works\andy0	AndyRuth
10	120	adventure-works\angela0	AngelaBarbariol
11	189	adventure-works\anibal0	AnibalSousa
12	264	adventure-works\annette0	AnnetteHill
13	57	adventure-works\annik0	AnnikStahl
14	198	adventure-works\arvind0	ArvindRao
15	44	adventure-works\ascott0	A. ScottWright

Funkcje definiowane przez użytkownika

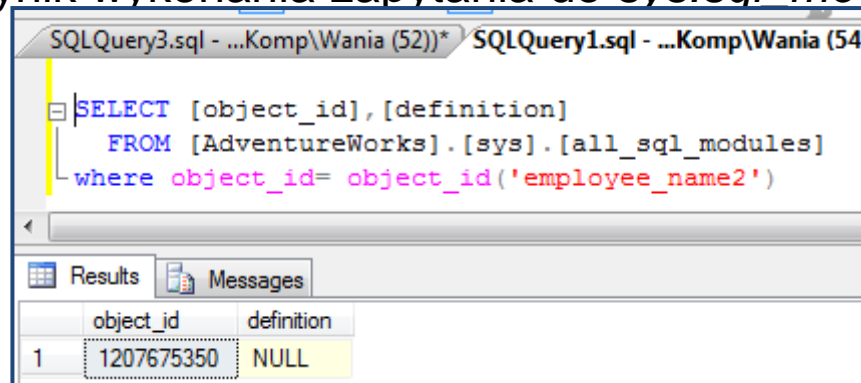
Przykład funkcji skalarnej – opcja ENCRYPTION

Strona 9

Tworzenie funkcji z opcją ENCRYPTION

```
USE AdventureWorks;
go
CREATE FUNCTION dbo.employee_name2(@employeeid AS INT)
RETURNS VARCHAR(100)
WITH ENCRYPTION
...
```

Wynik wykonania zapytania do sys.sql_modules



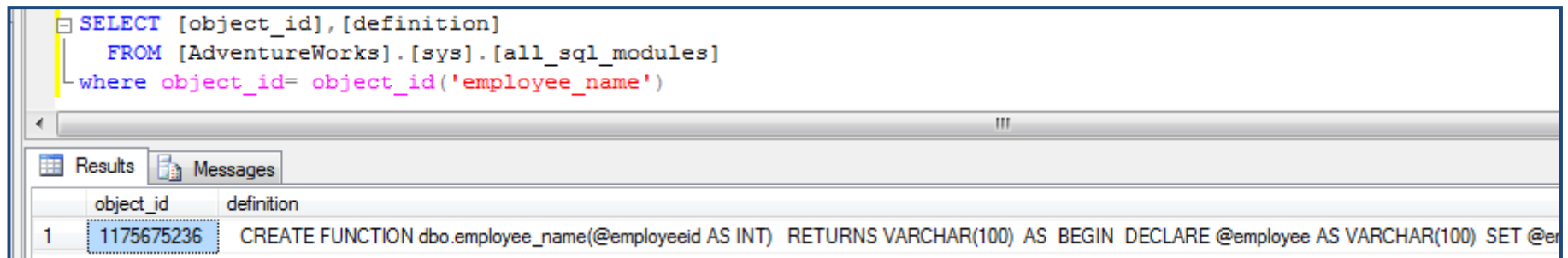
The screenshot shows a SQL query window with the following query:

```
SELECT [object_id], [definition]
FROM [AdventureWorks].[sys].[all_sql_modules]
where object_id= object_id('employee_name2')
```

The Results tab is active, displaying a single row:

	object_id	definition
1	1207675350	NULL

Wynik wykonania tego zapytania dla analogicznej funkcji bez opcji ENCRYPTION:



The screenshot shows a SQL query window with the following query:

```
SELECT [object_id], [definition]
FROM [AdventureWorks].[sys].[all_sql_modules]
where object_id= object_id('employee_name')
```

The Results tab is active, displaying a single row:

	object_id	definition
1	1175675236	CREATE FUNCTION dbo.employee_name(@employeeid AS INT) RETURNS VARCHAR(100) AS BEGIN DECLARE @employee AS VARCHAR(100) SET @e

Funkcje definiowane przez użytkownika

Wywoływanie skalnych funkcji UDF w zapytaniach stanowi duże obciążenie, zwłaszcza w przypadku gdy parametrami wejściowymi funkcji są atrybuty zewnętrznej tabeli.

Proces wywoływania funkcji dla wszystkich zwracanych wierszy, nawet jeżeli jest to tylko funkcja skalarna z prostą operacją arytmetyczną i klauzula RETURN, znacząco obniża efektywność tego zapytania.

Przykład

Zadanie: Z tabeli towar chcemy wyświetlić ceny wszystkich towarów zwiększony np. o domyślną wartość podatku VAT (22%).

Zalecenie:

Wydajniej jest robić to z za pomocą wyrażenia wbudowanego

```
SELECT cena+cena*22/100, ... FROM TOWAR
```

, niż za pomocą zdefiniowanej funkcji UDF, np.:

```
SELECT cena_z_VAT(cena), ... FROM TOWAR
```

Funkcje definiowane przez użytkownika

Tabelaryczne wbudowane funkcje UDF

Strona 11

Tabelaryczne funkcje UDF są funkcjami zwracającymi dane w postaci tabeli i zazwyczaj są stosowane w klauzuli *FROM* zewnętrznego zapytania.

Wbudowane tabelaryczne funkcje UDF podobnie jak widoki zwracają tabelę definiowaną przy pomocy zapytania, z tą różnicą, że możemy określić parametry wejściowe – argumenty funkcji.

Wbudowana tabelaryczna funkcja w odróżnieniu od złożonych tabelarycznych funkcji UDF nie zawiera bloku *BEGIN/END*, a tylko klauzulę *RETURN* z zapytaniem T-SQL.

Przykład wbudowanej tabelarycznej funkcji UDF:

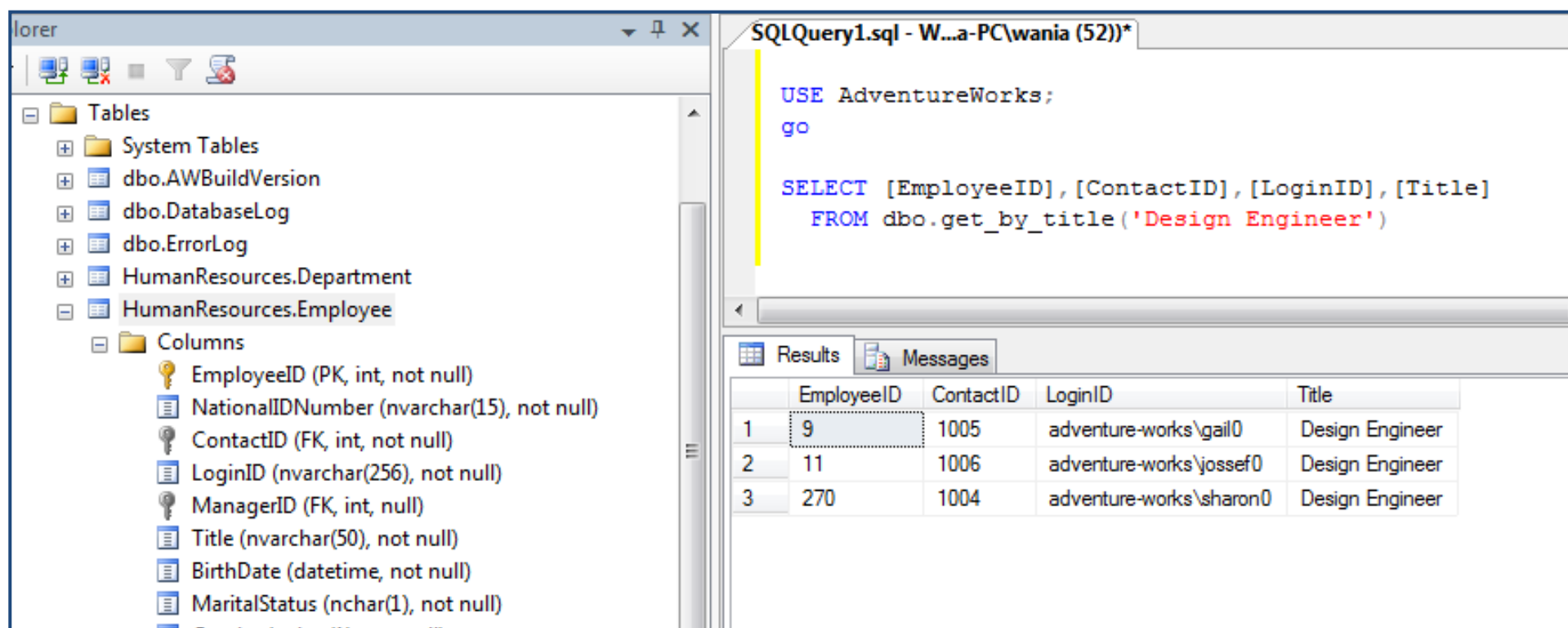
```
USE AdventureWorks;
go
CREATE FUNCTION dbo.get_by_title (@title as varchar(25))
RETURNS TABLE
AS
RETURN
SELECT [EmployeeID],[ContactID],[LoginID],[Title]
FROM [AdventureWorks].[HumanResources].[Employee]
WHERE Title=@title
```

Funkcje definiowane przez użytkownika

Tabelaryczne wbudowane funkcje UDF

Strona 12

Przykład wywołania utworzonej wbudowanej tabelarycznej funkcji UDF:



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing the 'HumanResources.Employee' table. The 'Columns' folder is also expanded, listing the following fields: EmployeeID (PK, int, not null), NationalIDNumber (nvarchar(15), not null), ContactID (FK, int, not null), LoginID (nvarchar(256), not null), ManagerID (FK, int, null), Title (nvarchar(50), not null), BirthDate (datetime, not null), and MaritalStatus (nchar(1), not null). On the right, the 'SQLQuery1.sql' window shows the following T-SQL code:

```
USE AdventureWorks;
go

SELECT [EmployeeID], [ContactID], [LoginID], [Title]
FROM dbo.get_by_title('Design Engineer')
```

Below the query window, the 'Results' tab is active, displaying the output of the query as a table with 5 columns: EmployeeID, ContactID, LoginID, and Title. The results show three rows of data for employees with the title 'Design Engineer'.

	EmployeeID	ContactID	LoginID	Title
1	9	1005	adventure-works\gail0	Design Engineer
2	11	1006	adventure-works\jossef0	Design Engineer
3	270	1004	adventure-works\sharon0	Design Engineer

Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 13

- Tabelaryczne złożone funkcje UDF zwraca zmienną tabelaryczną i zawiera wiele instrukcji T-SQL.
- Stosowane są gdy nie można w jednym zapytaniu zawrzeć wyników dla zwracanej przez funkcję tabeli.
- SQL Server wbudowane funkcje tabelaryczne UDF traktuje podobnie jak widoki, natomiast złożone tabelaryczne funkcje UDF traktowane są w sposób zbliżony do procedur składowanych.
- Pomimo tego złożone tabelaryczne funkcje UDF nie umożliwiają dokonywania modyfikacji i można je umieszczać jedynie w klauzuli FROM zapytania SELECT.

Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 14

Założenia dla przykładowej złożonej tabelarycznej funkcji UDF

(dla testowej bazy *AdventureWorks*):

- Utworzona złożona funkcja tabelaryczna będzie miała za zadanie pobranie informacji o fladze wynagrodzeń dla pracowników w danym wybranym kraju (argument funkcji).
- Funkcja powinna zwrócić informacje o liczbie pracowników w regionach tego kraju z odpowiednią wartością flagi wynagrodzeń (wartość 0 lub 1).
- Funkcja pobierze dane o wszystkich regionach w danym kraju, następnie dla każdego regionu sprawdzi liczbę pracowników z odpowiednimi wartościami flagi.
- Przed zwróceniem przez funkcję danych, powinny być usunięte dane o regionach z nieprzypisanymi pracownikami z odpowiednimi wartościami flagi wynagrodzeń

Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 15

Kod T-SQL przykładowej złożonej tabelarycznej funkcji UDF:

```
CREATE FUNCTION dbo.get_employee_by_coutry (@country as varchar(50))
RETURNS @stat TABLE (Country nchar(50), StateProvinceName nvarchar(50), salaried int, notsalaried int)
AS
BEGIN
--deklaracja zmiennych pomocniczych
DECLARE @SP nvarchar(50), @salaried int, @notsalaried int
--utworzenie kursora zawierającego wszystkie regiony dla wybranego kraju
DECLARE cur CURSOR FOR
SELECT SP.Name from [AdventureWorks].[Person].[StateProvince] SP,
               [AdventureWorks].[Person].[CountryRegion] CR
WHERE SP.CountryRegionCode=CR.[CountryRegionCode]
      and CR.[Name]=@country
--otwarcie kursora
OPEN cur
--pobranie pierwszej wartości dla kursora
FETCH NEXT FROM cur INTO @SP
...
```

Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 16

Kod T-SQL przykładowej złożonej tabelarycznej funkcji UDF(cd.):

```
--pętla wykonywana dla wszystkich regionów dla wybranego kraju
WHILE @@FETCH_STATUS = 0
BEGIN
--pobranie informacji o liczbie pracowników dla tego region z wartością dla flagi wynagrodzenia = 1
    SELECT @salaried=count(*) FROM [AdventureWorks].[Person].[Address] AD,
        [AdventureWorks].[HumanResources].[Employee] EM,
        [AdventureWorks].[HumanResources].[EmployeeAddress] EA,
        [AdventureWorks].[Person].[StateProvince] SP
    WHERE EM.employeeid=EA.employeeid and EA.addressid=AD.addressid
        and SP.stateprovinceid=AD.stateprovinceid
        and SP.name=@SP and EM.SalariedFlag=1
--pobranie informacji o liczbie pracowników dla tego region z wartością dla flagi wynagrodzenia = 0
    SELECT @notsalaried=count(*) FROM [AdventureWorks].[Person].[Address] AD,
        [AdventureWorks].[HumanResources].[Employee] EM,
        [AdventureWorks].[HumanResources].[EmployeeAddress] EA,
        [AdventureWorks].[Person].[StateProvince] SP
    WHERE EM.employeeid=EA.employeeid and EA.addressid=AD.addressid
        and SP.stateprovinceid=AD.stateprovinceid
        and SP.name=@SP and EM.SalariedFlag=0
    ...

```


Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 17

Kod T-SQL przykładowej złożonej tabelarycznej funkcji UDF(cd.):

```
-- wstawienie do zmiennej tabelarycznej pobranych wartości
INSERT INTO @stat VALUES (@country,@SP,@salaried,@notsalaried)
-- pobranie kolejnej wartości kursora
FETCH NEXT FROM cur INTO @SP
-- zakończenie pętli kursora pobierającej informacje dla pracowników z danego regionu
END

...
--zamknięcie kursora
CLOSE cur
--zwolnienie pamięci
DEALLOCATE cur

-- usunięcie informacji o regionach bez pracowników
DELETE FROM @stat WHERE salaried=0 and notsalaried=0

--zwrócenie przez funkcję zadeklarowanej zmiennej tabelarycznej
RETURN;
-- zakończenie funkcji
END
```

Funkcje definiowane przez użytkownika

Tabelaryczne złożone funkcje UDF

Strona 18

Testowanie opracowanej przykładowej złożonej tabelarycznej funkcji UDF:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'AdventureWorks' database is expanded, showing the 'HumanResources.Employee' table. The table structure is detailed in the 'Columns' pane, listing fields like EmployeeID (PK), NationalIDNumber, ContactID (FK), LoginID, ManagerID (FK), Title, and BirthDate. On the right, a SQL query is executed in a query window:

```
use AdventureWorks;
go
select * from dbo.get_employee_by_country ('United States')
```

The 'Results' pane shows the output of the query, which is a table with 5 columns: Country, StateProvinceName, salaried, and notsalaried. The data is as follows:

	Country	StateProvinceName	salaried	notsalaried
1	United States	California	2	0
2	United States	Massachusetts	1	0
3	United States	Michigan	1	0
4	United States	Minnesota	1	1
5	United States	Oregon	1	0
6	United States	Tennessee	1	0
7	United States	Utah	1	0
8	United States	Washington	38	237

Funkcje definiowane przez użytkownika

Podsumowanie

Strona 19

- Funkcje UDF mogą być umieszczane w:
 - zapytaniach
 - ograniczeniach
 - kolumnach obliczeniowych
 - ...
- Można budować skalarne i tabelaryczne funkcje obliczeniowe.
- Funkcje nie mogą wpływać na stan bazy danych poza ich obszarem, nie mogą modyfikować danych.
- W SQL Server możemy opisywać funkcję w T-SQL oraz w językach platformy .NET
- Można w nich efektywnie implementować:
 - logikę proceduralną
 - złożone obliczenia
 - przetwarzanie ciągów
 - ...

Procedury składowane

Charakterystyka

Strona 20

- Procedury składowane to obiekty programistyczne uruchamiane po stronie serwera baz danych.
- W przeciwieństwie do funkcji mogą modyfikować dane oraz modyfikować obiekty.
- Możliwe jest wprowadzenie dodatkowej warstwy bezpieczeństwa, gdyż można użytkownikom bazy danych nadać uprawnienia do procedur składowanych zamiast do obiektów docelowych, które będą przez te procedury wykorzystywane.
- Dzięki hermetyzacji kodu możliwe jest modyfikowanie procedur przy niezmiennym interfejsie w sposób niezauważalny dla użytkowników bazy danych, w przeciwieństwie do przypadku gdy kod biznesowy przechowywany jest po stronie klienta.
- Właściwe wykorzystanie powoduje wzrost efektywności poprzez wielokrotne wykorzystanie tego samego planu wykonywania zapytań oraz zmniejszenie ruchu pomiędzy w sieci pomiędzy klientem i serwerem.

Rodzaje procedur składowanych:

- Definiowane przez użytkownika(UDF)
 - opisywane z wykorzystaniem Transact-SQL
 - opisywane z wykorzystaniem języka CLR
 - Specjalne procedury składowane (*sp_...*)
- Systemowe procedury składowane
- Rozszerzone procedury składowane – tworzenie zewnętrznych funkcji np. w języku C z wykorzystaniem API Extended Stored Procedure (w postaci plików *.dll)
- Tymczasowe procedury składowane – nazwy procedur poprzedzone znakami # lub ##

Procedury składowane

Definiowane przez użytkownika procedury T-SQL

Strona 22

Procedura UDF tworzona jest w bazie użytkownika z wykorzystaniem polecenia *CREATE PROCEDURE ...* i zazwyczaj współpracuje z obiektami tej bazy.

Procedurę UDF wykonuje z wykorzystaniem polecenia EXEC (EXECUTE) z podaniem nazwy wraz ze schematem oraz argumentem.

Przykładowe procedury T-SQL definiowane przez użytkownika dla testowej bazy danych *AdventureWorks*:

- Procedura modyfikująca przykładowe dane (*Product_cost_modification*)
- Procedura demonstrująca wprowadzenie dodatkowej warstwy bezpieczeństwa (*Show_product_price*) poprzez dodanie możliwości przeglądania ceny produktu, dla użytkownika, który nie ma dostępu do tabeli *Production.Product*

Procedury składowane

Procedury UDF T-SQL – modyfikacja danych

Strona 23

Tworzenie procedury *Product cost modification*:

The screenshot displays the Microsoft SQL Server Enterprise Manager interface. On the left, the 'Production.Product' table structure is shown with the following columns:

- ProductID (PK, int, not null)
- Name (Name(nvarchar(50)), not null)
- ProductNumber (nvarchar(25), not null)
- MakeFlag (Flag(bit), not null)
- FinishedGoodsFlag (Flag(bit), not null)
- Color (nvarchar(15), null)
- SafetyStockLevel (smallint, not null)
- ReorderPoint (smallint, not null)
- StandardCost (money, not null)
- ListPrice (money, not null)
- Size (nvarchar(5), null)
- SizeUnitMeasureCode (FK, nchar(3), null)
- WeightUnitMeasureCode (FK, nchar(3), null)
- Weight (decimal(8,2), null)
- DaysToManufacture (int, not null)

On the right, the SQL Query window shows the execution of a T-SQL procedure named 'Product_cost_modification'. The code is as follows:

```
USE AdventureWorks
GO
CREATE PROCEDURE dbo.Product_cost_modification
@prod_num AS nvarchar(25)=NULL,
@percent AS decimal(5,2)=0.0
AS
BEGIN
UPDATE Production.Product
SET
StandardCost+=StandardCost*@percent/100,
ListPrice+=ListPrice*@percent/100
WHERE Production.Product.ProductNumber=@prod_num
END
```

The Results pane at the bottom indicates: 'Command(s) completed successfully.'

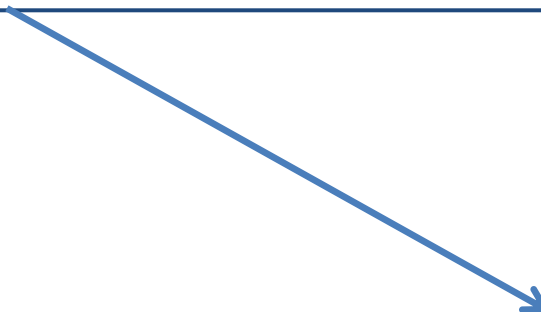
Procedury składowane

Procedury UDF T-SQL – modyfikacja danych

Strona 24

Testowanie procedury *Product cost modification*:

```
SQLQuery4.sql - ...Komp\Wania (56)) SQLQuery3.sql - ...Komp\Wania (54))* SQLQuery
USE AdventureWorks
GO
SELECT StandardCost, ListPrice FROM Production.Product
WHERE ProductNumber='BB-7421'
GO
EXEC dbo.Product_cost_modification 'BB-7421', 10
GO
SELECT StandardCost, ListPrice FROM Production.Product
WHERE ProductNumber='BB-7421'
```



Results	
StandardCost	ListPrice

23,9716	53,99
(1 row(s) affected)	
(1 row(s) affected)	
StandardCost	ListPrice

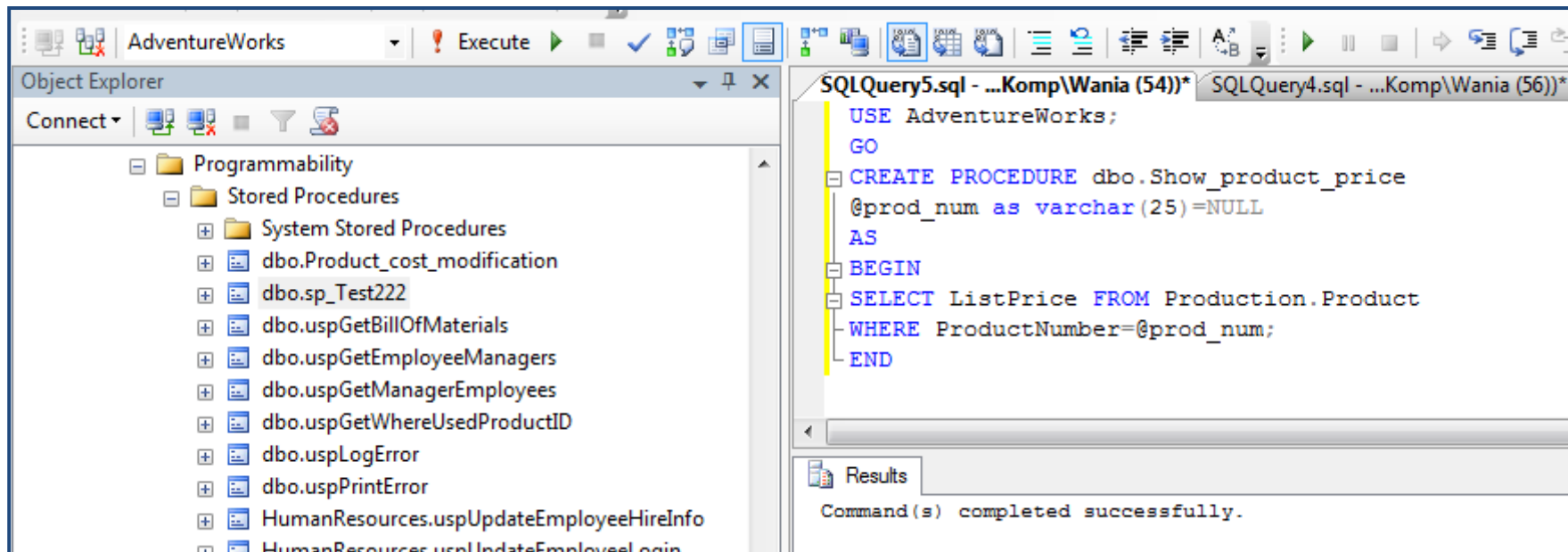
26,3688	59,389
(1 row(s) affected)	

Procedury składowane

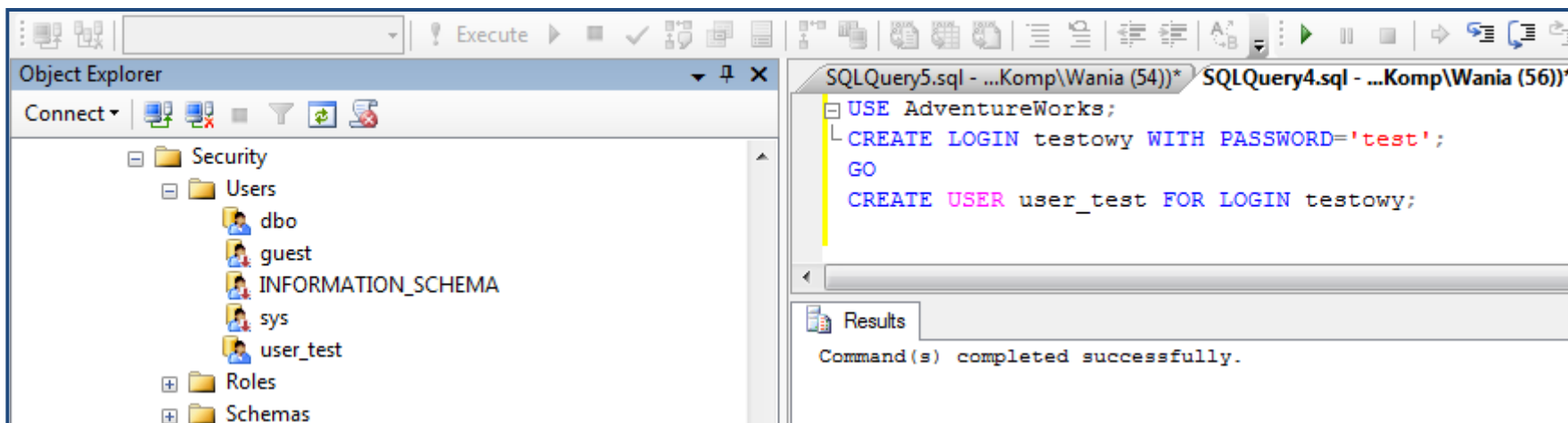
Procedury UDF T-SQL – dodatkowa warstwa bezpieczeństwa

Strona 25

Tworzenie procedury składowanej do wyświetlania ceny produktu:



Tworzenie testowego użytkownika:

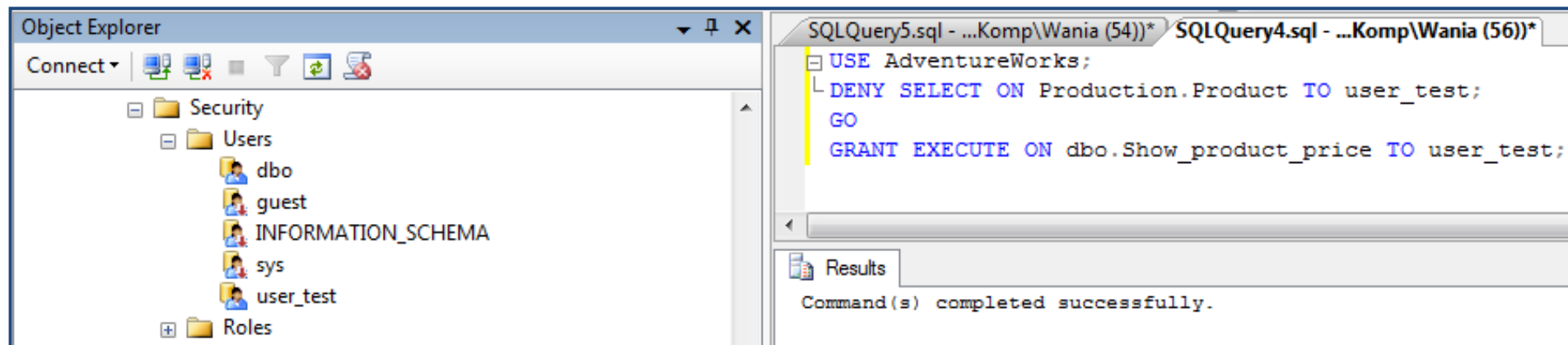


Procedury składowane

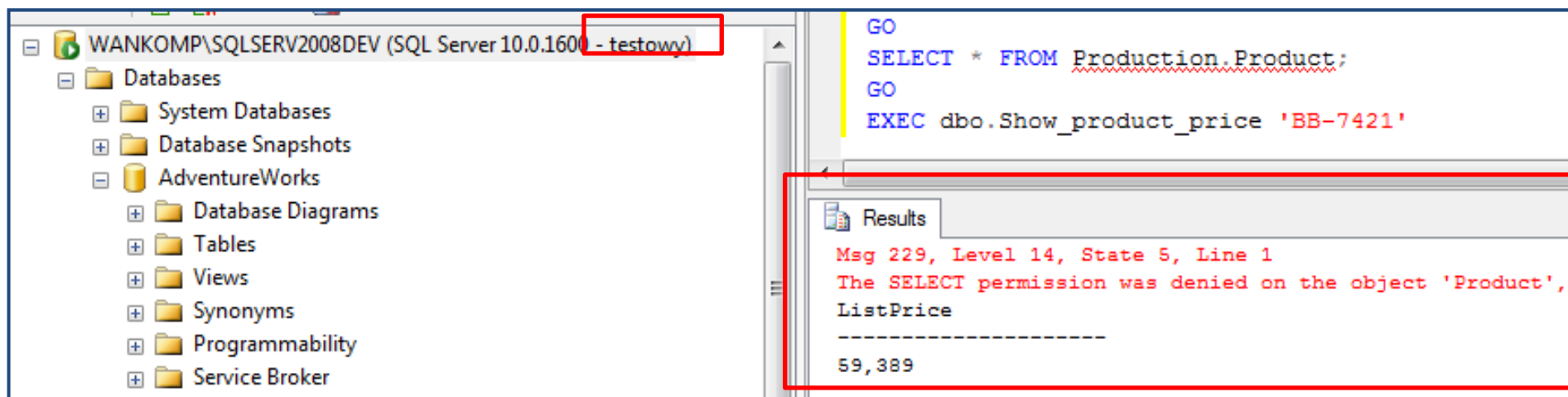
Procedury UDF T-SQL – dodatkowa warstwa bezpieczeństwa

Strona 26

Odebranie i nadanie testowemu użytkownikowi odpowiednich uprawnień:



Testowanie dodatkowej warstwy bezpieczeństwa:



Procedury składowane

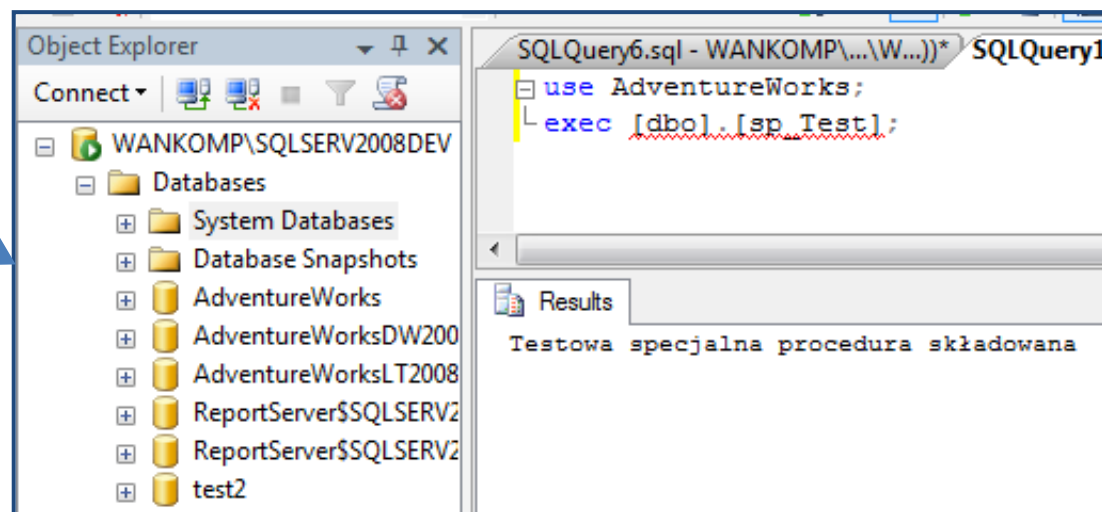
Specjalne procedury składowane

Strona 27

- Specjalne procedury składowane utworzone zostały w bazie *master* z nazwą rozpoczynającą się od ciągu *sp_...*
- Podczas jej wykonywanie nie musimy określać nazwy bazy danych w której jest ona przechowywana.
- Microsoft nie zaleca tworzenia własnych procedur z przedrostkiem *sp_*, gdyż jest on wykorzystywany do identyfikowania procedur systemowych

```
SQLQuery1.sql - WANKOMP\...\W...)*
USE master;
GO
CREATE PROC sp_Test
AS
Print 'Testowa specjalna
procedura składowana.'
GO
```

Query executed successfully.



- Procedury składowane to jedno z najbardziej zaawansowanych narzędzi SQL Server.
- Umiejętne stosowanie procedur powoduje tworzenie bezpiecznych baz danych i aplikacji o wysokiej efektywności.
- Stanowią dodatkową warstwę zabezpieczeń.
- Umożliwiają hermetyzację kodu.
- Redukują ruch w sieci.
- Dają możliwość wielokrotnego wykorzystywania planów realizacji
- SQL Server wspiera również procedury CLR co powoduje rozszerzenie ich funkcjonalności.

Wyzwalacze

Charakterystyka ogólna

Strona 29

- Wyzwalacze (ang. Trigger) są automatycznie uruchomianymi obiektami serwera bazy danych uruchomianymi w wyniku wystąpienia określonego zdarzenia na serwerze.
- Przez serwer traktowane są podobnie do procedur składowanych, jednak nie posiadają interfejsu i nie można ich wywoływać, gdyż są uruchamiane automatycznie.
- Wyzwalacze podobnie jak polecenia Transact-SQL stanowią część transakcji, tzn jeżeli zostanie wycofana transakcja poleceniem ROLLBACK to wycofane zostaną również polecenia wykonane w wyzwalaczu.
- Wyzwalacze umożliwiają automatyzację odpowiedzi dla określonych instrukcji bądź zdarzeń na serwerze.
- Mogą posłużyć do wymuszania zaawansowanych reguł integralności danych.

Rodzaje wyzwalaczy występujące w MS SQL Server 2008:

- Wyzwalacze dla grupy instrukcji DDL (ang. Data Definition Language), np. CREATE.... (T-SQL)
- Wyzwalacze dla grup poleceń DML (ang. Data Modification Language), np.: UPDATE ... (T-SQL)
- Wyzwalacze dla operacji logowania.

Podział ze względu na moment wykonania instrukcji wyzwalacza:

- AFTER – po wystąpieniu określonego zdarzenia
- INSTEAD OF – zastępują oryginalną instrukcję zdarzenia

UWAGA: W MS SQL Server 2008 nie występują wyzwalacze dla polecenia SELECT oraz wykonywane w trybie BEFORE!

- Podczas tworzenia wyzwalaczy AFTER istnieje możliwość wykorzystywania tabeli specjalnych *INSERTED* i *DELETED*.
- Tabele te zawierają odpowiednie dane dla wykonanych poleceń Transact-SQL, jak: INSERT, UPDATE i DELETE.
- Tabela *INSERTED* zawiera dane wprowadzone w instrukcjach INSERT oraz UPDATE, natomiast tabela *DELETED* zawiera dane usunięte poleceniem DELETE oraz nadpisane dane („stare wartości”) poleceniem UPDATE.
- Tabela *DELETED* pozostaje pusta dla poleceń typu INSERT, analogicznie jak tabela *INSERTED* dla poleceń typu DELETE.
- Obie tabele umożliwiają sprawne operacje na danych przy różnego rodzaju wyzwalaczach.

Wyzwalacze

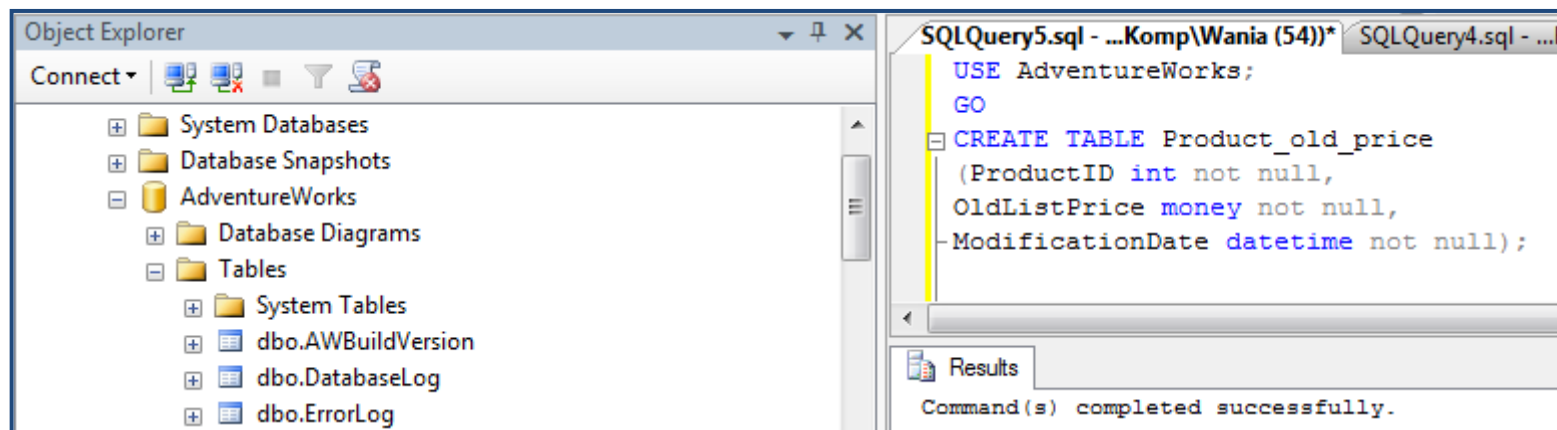
Przykład wykorzystania tabeli specjalnych

Strona 32

Przykładowy wyzwalacz będzie wprowadzał do dodatkowej tabeli starą wartość ceny produktu przed dokonaniem modyfikacji.

Podczas uruchomienia wyzwalacza po operacji UPDATE zostanie sprawdzona z wykorzystaniem tabel *INSERTED* i *DELETED* stara i nowa wartość ceny produktu. Informacja do dodatkowej tabeli zawierającej starą cenę powinna zostać wprowadzona tylko w przypadku, gdy modyfikacji uległa cena produktu.

Utworzenie dodatkowej tabeli przechowującej stare wartości ceny produktu:

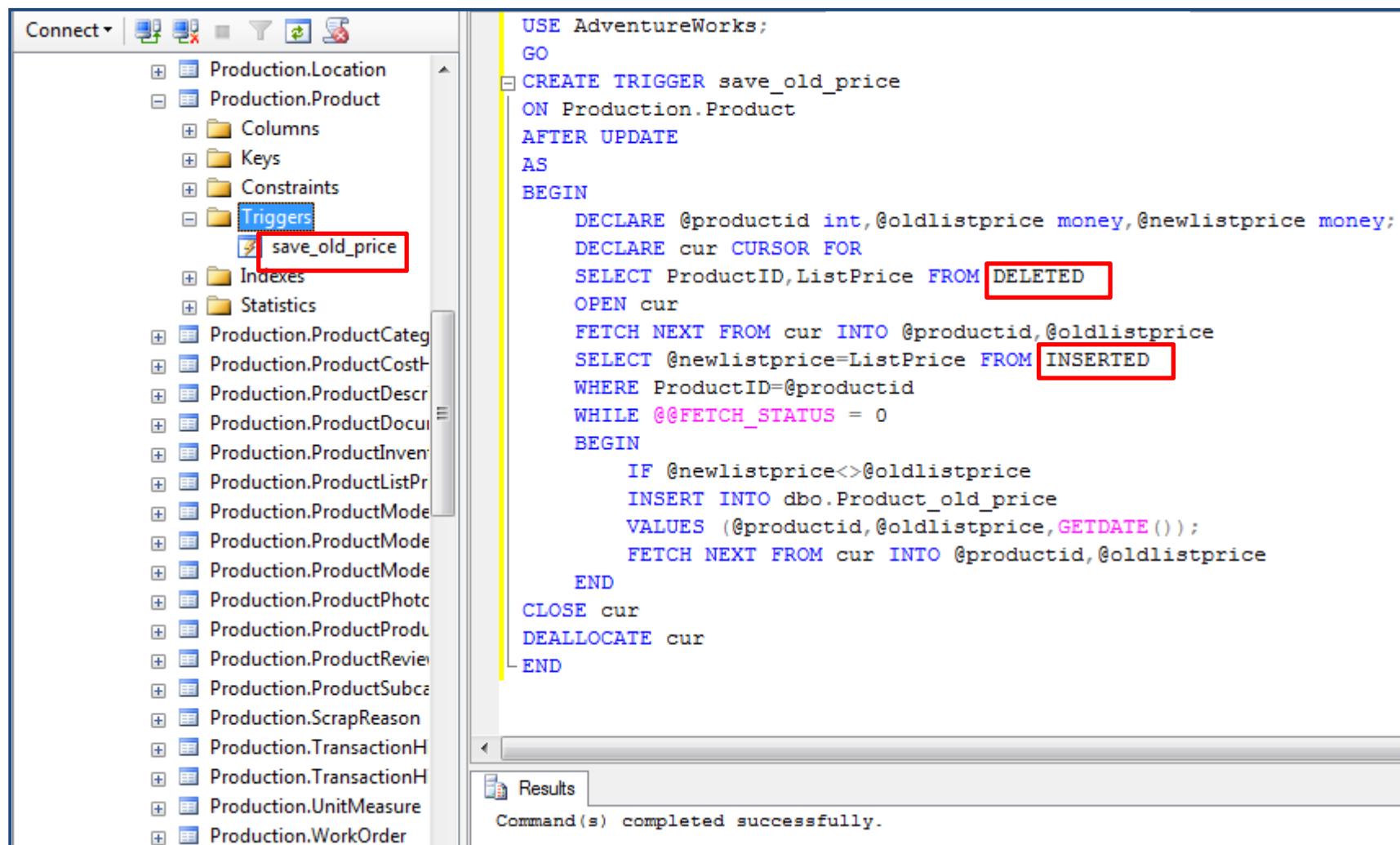


Wyzwalacze

Przykład wykorzystania tabeli specjalnych

Strona 33

Utworzenie założonego wyzwalacza dla tabeli *Production.Product* testowej bazy *AdventureWorks*:



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Production.Triggers' folder is expanded, and the 'save_old_price' trigger is highlighted with a red box. The right pane shows the SQL script for creating this trigger:

```
USE AdventureWorks;
GO
CREATE TRIGGER save_old_price
ON Production.Product
AFTER UPDATE
AS
BEGIN
    DECLARE @productid int, @oldlistprice money, @newlistprice money;
    DECLARE cur CURSOR FOR
    SELECT ProductID, ListPrice FROM DELETED
    OPEN cur
    FETCH NEXT FROM cur INTO @productid, @oldlistprice
    SELECT @newlistprice = ListPrice FROM INSERTED
    WHERE ProductID = @productid
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @newlistprice <> @oldlistprice
            INSERT INTO dbo.Product_old_price
            VALUES (@productid, @oldlistprice, GETDATE());
        FETCH NEXT FROM cur INTO @productid, @oldlistprice
    END
    CLOSE cur
    DEALLOCATE cur
END
```

The 'DELETED' and 'INSERTED' table references in the SQL script are highlighted with red boxes. At the bottom, the 'Results' pane shows the message: 'Command(s) completed successfully.'

Wyzwalacze

Przykład wykorzystania tabeli specjalnych

Strona 34

Sprawdzenie działania utworzonego wyzwalacza:

```
SQLQuery7.sql - ...Komp\Wania (56))* SQLQuery6.sql - ...Komp
USE AdventureWorks;
GO
SELECT ListPrice FROM Production.Product
WHERE ProductID BETWEEN 997 AND 999
GO
SELECT * FROM dbo.Product_old_price
GO
UPDATE Production.Product
SET ModifiedDate=GETDATE()
WHERE ProductID=997;
UPDATE Production.Product
SET ListPrice+=ListPrice*3/100
WHERE ProductID BETWEEN 998 AND 999
GO
SELECT ListPrice FROM Production.Product
WHERE ProductID BETWEEN 997 AND 999
GO
SELECT * FROM dbo.Product_old_price
```

Results		
ListPrice		

539,99		
556,1897		
556,1897		
(3 row(s) affected)		
ProductID	OldListPrice	ModificationDate

(0 row(s) affected)		
(1 row(s) affected)		
(1 row(s) affected)		
(1 row(s) affected)		
(2 row(s) affected)		
ListPrice		

539,99		
572,8753		
572,8753		
(3 row(s) affected)		
ProductID	OldListPrice	ModificationDate

999	556,1897	2012-08-28 12:07:42.540
998	556,1897	2012-08-28 12:07:42.540
(2 row(s) affected)		

Analogiczny wyzwalacz można skonstruować z wykorzystaniem wbudowanej funkcji COLUMNS_UPDATED.

Funkcja ta zwraca ciąg binarny w którym każda kolumna modyfikowanej tabeli jest reprezentowana przez jeden bit. Poszczególne bity będą zawierały wartość 1, jeżeli odpowiednia kolumna została zmodyfikowana.

Bajty w ciągu binarnym uporządkowane są od lewej do prawej, natomiast bity w bajtach od prawej do lewej.

Do wyodrębnienia bajtu w którym znajduje się bit dla wybranego numeru kolumny możemy wykorzystać następujące wyrażenie:

```
SUBSTRING(COLUMNS_UPDATED(), (@nr_kolumny-1)/8+1, 1)
```

Natomiast kompletne wyrażenie dla sprawdzenia wartości bitu dla danego numeru kolumny ma postać:

```
IF SUBSTRING(COLUMNS_UPDATED(), (@nr_kolumny-1)/8+1, 1)  
& POWER(2, (@nr_kolumny-1)%8) > 0
```

Wyzwalacze

Sprawdzenie wbudowanej funkcji COLUMNS_UPDATED

Strona 36

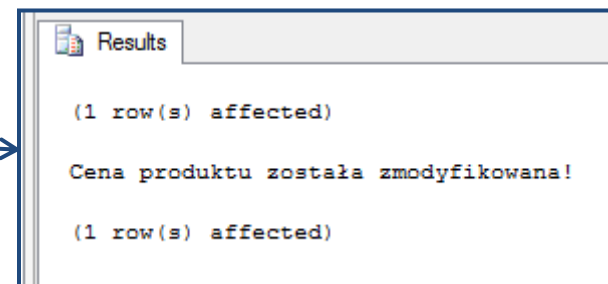
Dla sprawdzenia utworzono wyzwalacz zwracający informację czy podczas modyfikacji danych produktu wartość jego ceny została zmodyfikowana.

Tworzenie wyzwalacza:

```
USE AdventureWorks;
GO
CREATE TRIGGER test_price_update
ON Production.Product
AFTER UPDATE
AS
BEGIN
DECLARE @nr_kolumny int;
SET @nr_kolumny=10;
IF SUBSTRING(COLUMNS_UPDATED(), (@nr_kolumny-1)/8+1,1)
& POWER(2, (@nr_kolumny-1)%8)>0
PRINT 'Cena produktu została zmodyfikowana!'
END
```

Testowanie działania:

```
USE AdventureWorks;
GO
UPDATE Production.Product
SET ModifiedDate=GETDATE()
WHERE ProductID=997;
GO
UPDATE Production.Product
SET ListPrice+=ListPrice*3/100
WHERE ProductID=998;
```



Results

(1 row(s) affected)

Cena produktu została zmodyfikowana!

(1 row(s) affected)

Wyzwalacze

Zagnieżdżenie i rekurencja wyzwalaczy

Strona 37

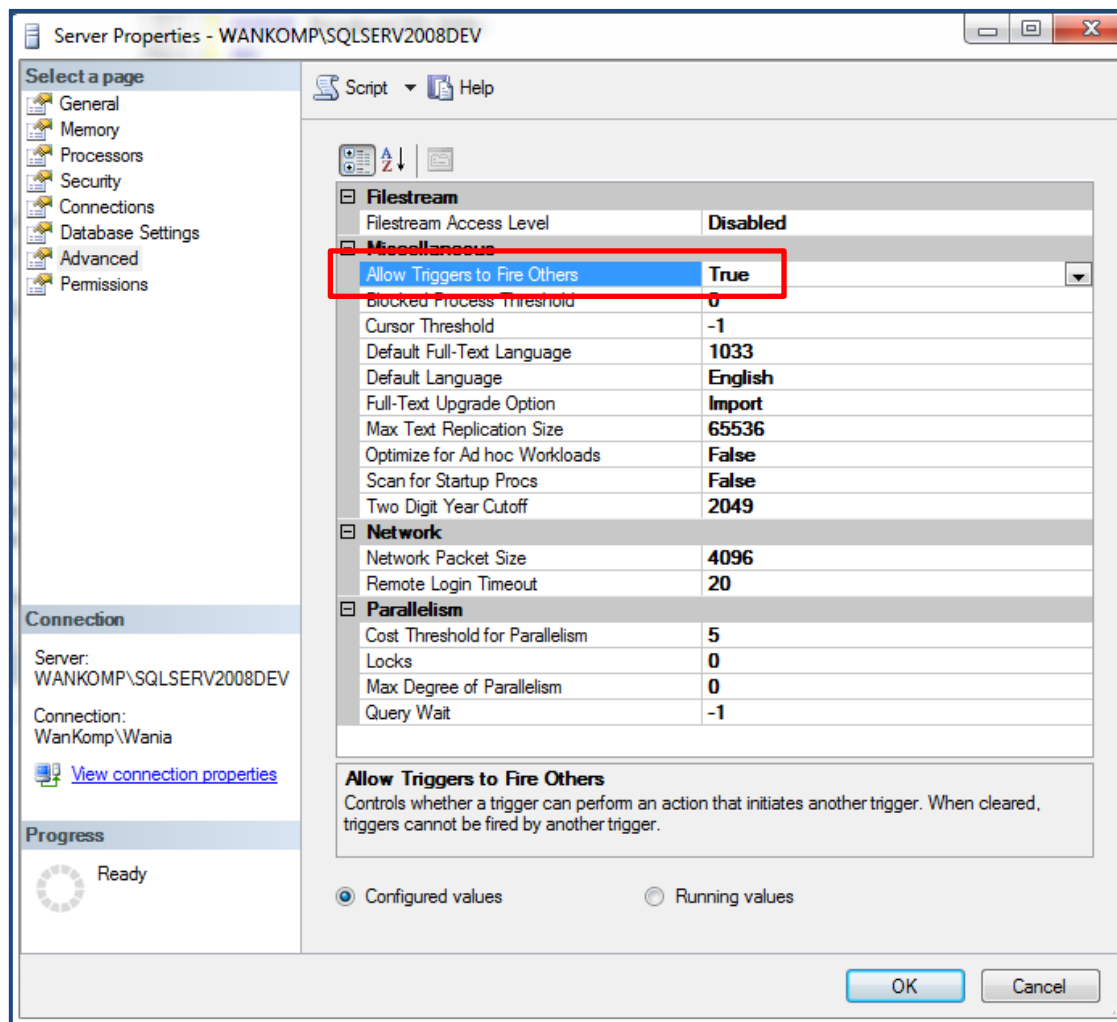
- SQL Serwer umożliwia zarówno zagnieżdżenie, jak i rekurencję wyzwalaczy.
- Zagnieżdżenie wyzwalaczy zachodzi, gdy akcja jednego wyzwalacza powoduje wyzwolenie innego.
- Zagnieżdżenie kontrolowane jest na poziomie serwera, poprzez wartość zmiennej *neted_triggers*
- Rekurencja zachodzi gdy akcja wykonywanego wyzwalacza powoduje jego ponowne wyzwolenie.
- Rekurencja kontrolowana jest na poziomie bazy danych poprzez ustawianie opcji bazy danych *RECURSIVE_TRIGGERS*
- Dla rekurencji Sql Server posiada stały limit zagnieżdżenia o wartości 32. Próba wykonania trigera na 33 poziomie rekurencji kończy się niepowodzeniem, a wszystkie operacje są anulowane.

Wyzwalacze

Definiowanie możliwości zagnieżdżenia i rekurencji wyzwalaczy

Strona 38

Modyfikowanie zmiennej definiującej możliwość zagnieżdżenia wyzwalaczy (domyślnie włączone):

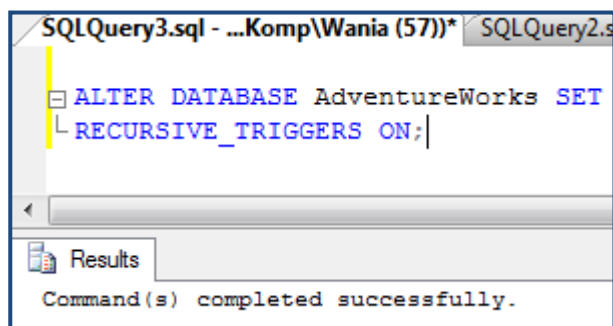


Wyzwalacze

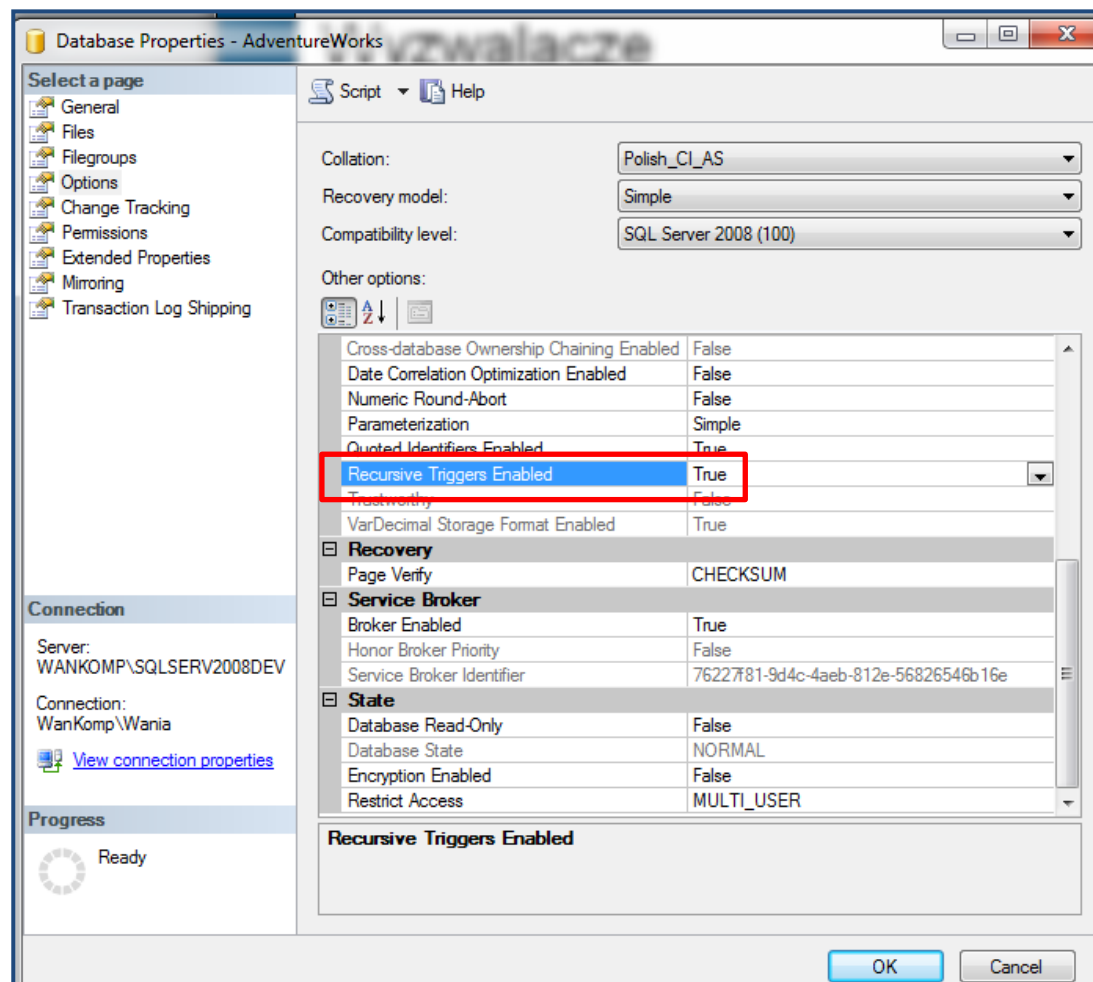
Definiowanie możliwości zagnieżdżenia i rekurencji wyzwalaczy

Strona 39

Przykład włączania opcji rekursji wyzwalaczy dla testowej bazy danych *AdventureWorks*:



lub



Wyzwalacze

Wyzwalacze INSTEAD OF

Strona 40

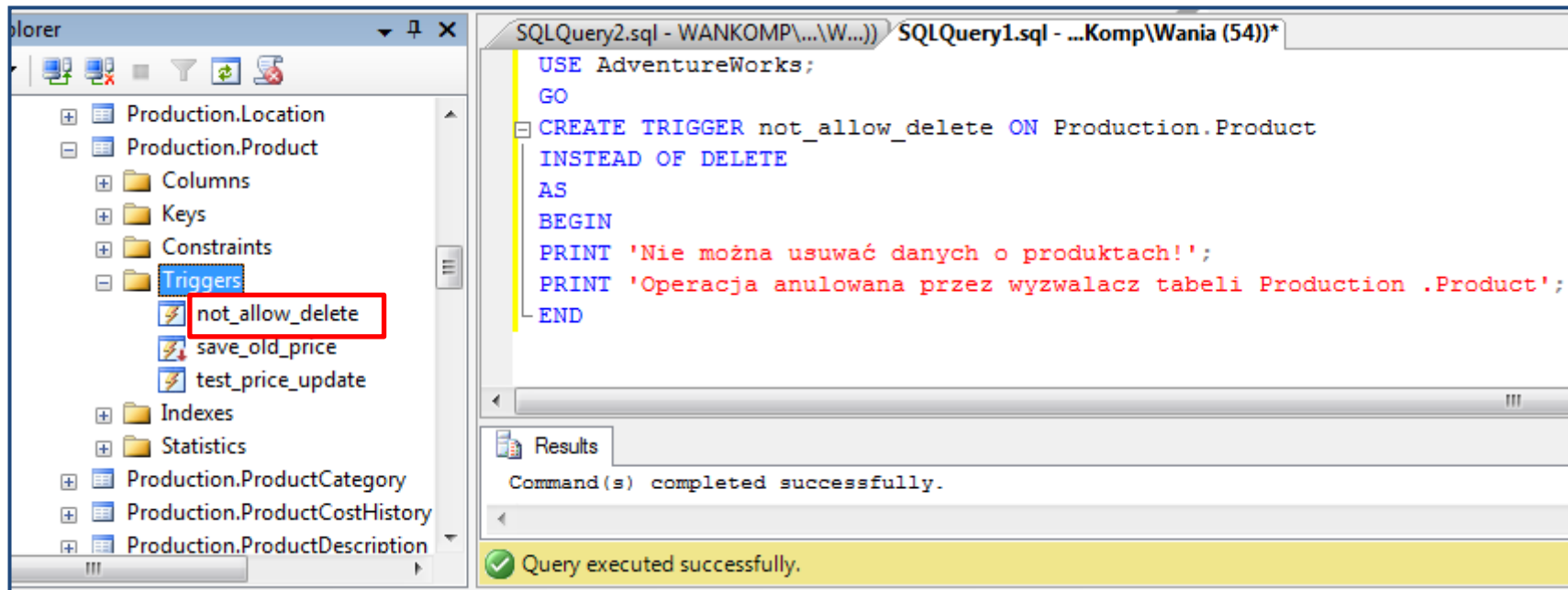
- Wyzwalacze tego rodzaju są uruchamiane zamiast oryginalnej wywołanej instrukcji dla wybranego obiektu docelowego.
- Oryginalna instrukcja nie jest wykonywana, natomiast w jej miejsce wykonywany jest kod wyzwalacza.
- Wyzwalacze tego typu można tworzyć również dla widoków.
- Są uruchamiane przed sprawdzeniem ograniczeń, czyli można wykonać ich kod nawet w przypadku gdyby oryginalna instrukcje nie mogła zostać wykonana.
- Wyzwalacza INSTEAD OF wspierają pośrednią rekurencję, tzn. mogą się nawzajem wyzwać.
- Wyzwalacze INSTEAD OF mogą stanowić dodatkowe zabezpieczenie, np. zabronić usunięcia, modyfikacji lub dodania danych czy obiektów bazy danych.

Wyzwalacze

Tworzenie przykładowego wyzwalacza INSTEAD OF

Strona 41

Zadaniem tworzonego wyzwalacza będzie wyświetlenie odpowiedniego komunikatu przy próbie wykonania operacji DELETE na tabeli *Production.Product* testowej bazy *AdventureWorks*

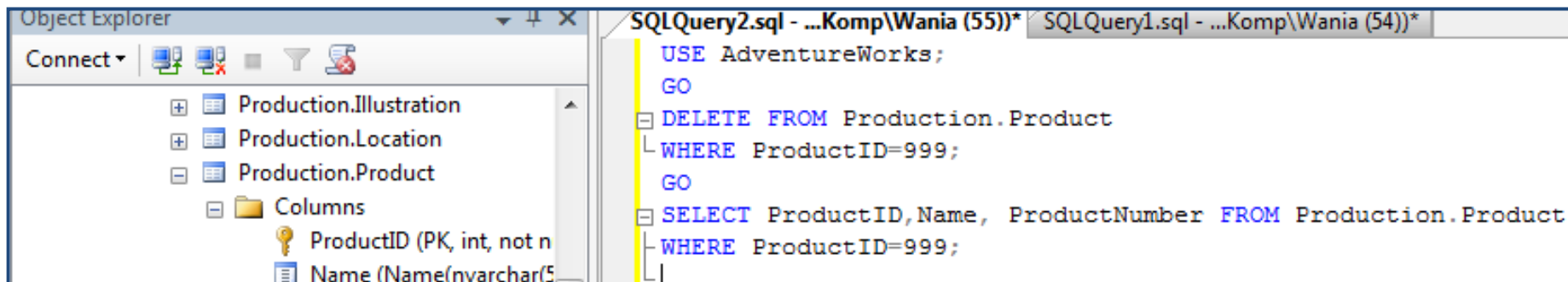


Wyzwalacze

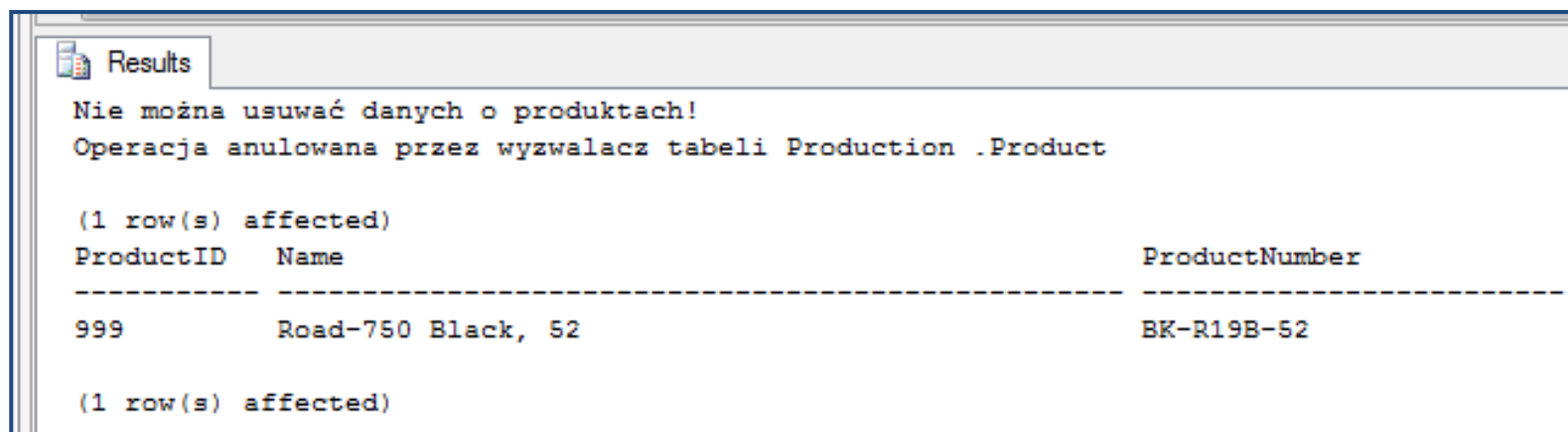
Testowanie utworzonego wyzwalacza INSTEAD OF

Strona 42

Wywołanie operacji DELETE:



Komunikat serwera bazy danych:



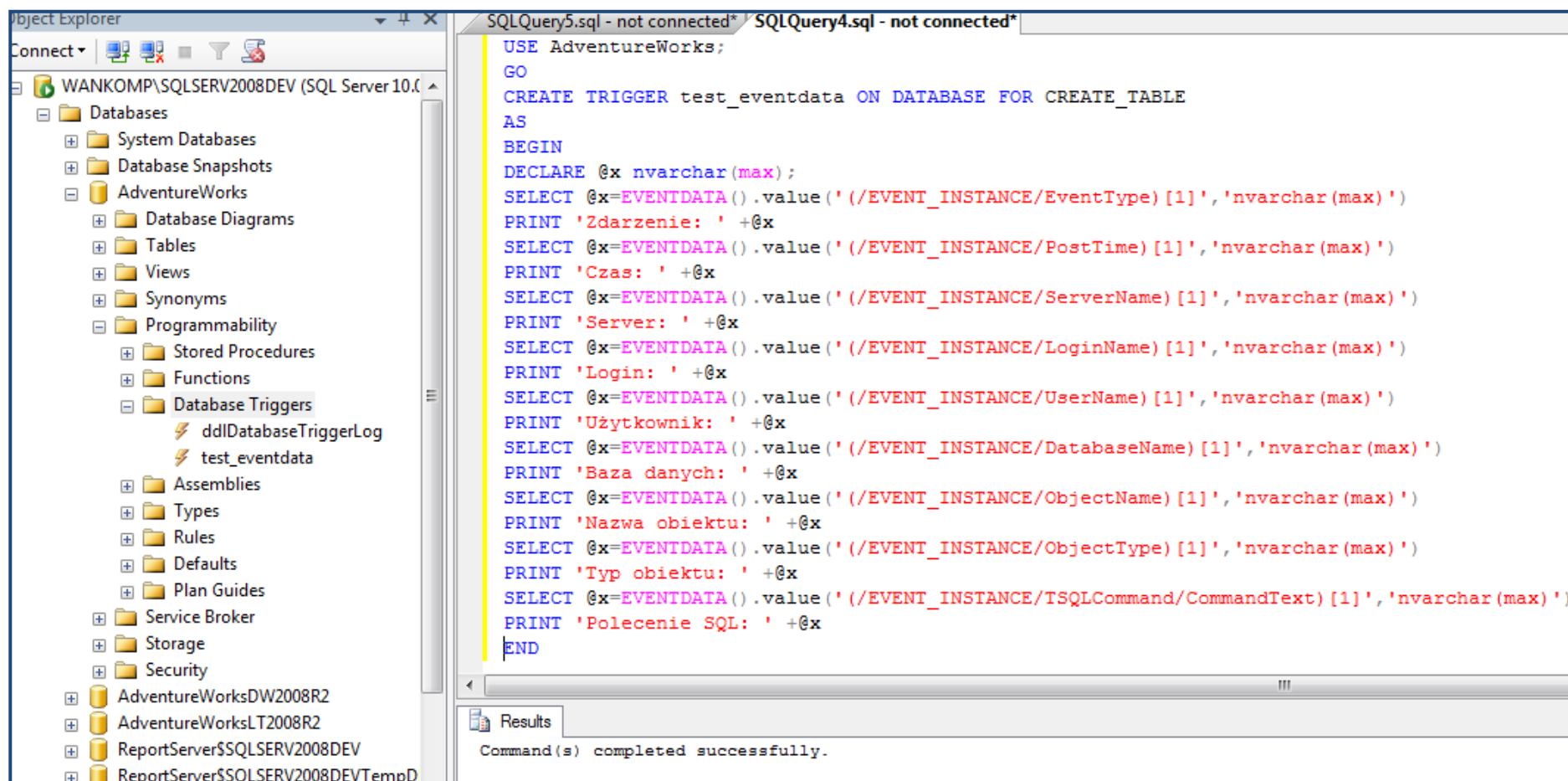
- Wyzwalacze DDL (Data Definition Language) służą do reagowania na określone zdarzenia na serwerze
- W SQL Server możliwe jest definiowanie wyzwalaczy DDL jedynie z opcją AFTER
- Mogą być tworzone na poziomie serwera lub na poziomie bazy danych
- Dostępna jest funkcja EVENTDATA pozwalająca uzyskać informację o zdarzeniu które wywołało wyzwalacz
- Aby wyzwalacz uniemożliwił przeprowadzenie modyfikacji na obiekcie należy wywołać polecenie ROLLBACK TRAN w kodzie wyzwalacza

Wyzwalacze

Wyzwalacze DDL – funkcja EVENTDATA

Strona 44

Utworzenie wyzwalacza DDL z funkcją EVENTDATA:



The screenshot displays the SQL Server Enterprise Manager interface on the left and a SQL Query window on the right. The Enterprise Manager shows the 'Database Triggers' folder expanded under the 'AdventureWorks' database, with a new trigger named 'test_eventdata' being created. The SQL Query window shows the following T-SQL code:

```
USE AdventureWorks;
GO
CREATE TRIGGER test_eventdata ON DATABASE FOR CREATE_TABLE
AS
BEGIN
DECLARE @x nvarchar(max);
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/EventType) [1]', 'nvarchar(max)')
PRINT 'Zdarzenie: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/PostTime) [1]', 'nvarchar(max)')
PRINT 'Czas: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/ServerName) [1]', 'nvarchar(max)')
PRINT 'Server: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/LoginName) [1]', 'nvarchar(max)')
PRINT 'Login: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/UserName) [1]', 'nvarchar(max)')
PRINT 'Użytkownik: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/DatabaseName) [1]', 'nvarchar(max)')
PRINT 'Baza danych: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/ObjectName) [1]', 'nvarchar(max)')
PRINT 'Nazwa obiektu: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/ObjectTypeId) [1]', 'nvarchar(max)')
PRINT 'Typ obiektu: ' +@x
SELECT @x=EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText) [1]', 'nvarchar(max)')
PRINT 'Polecenie SQL: ' +@x
END
```

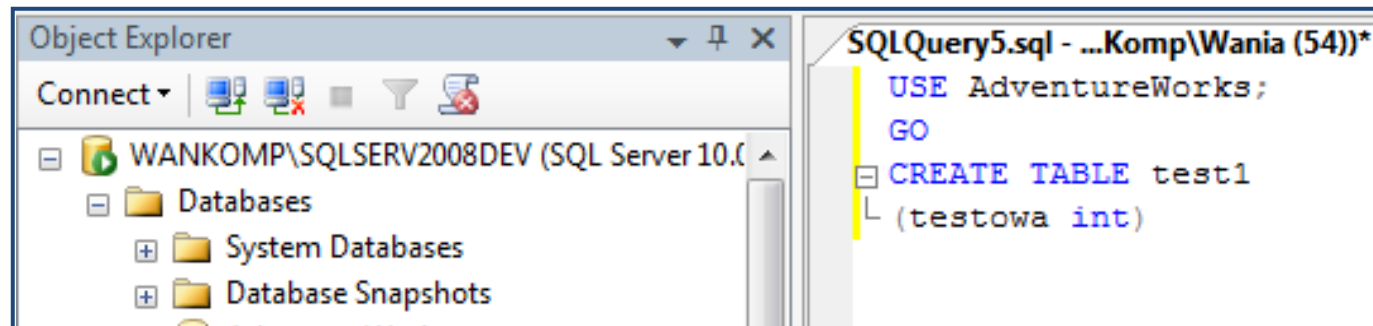
The bottom of the SQL Query window shows the 'Results' pane with the message: 'Command(s) completed successfully.'

Wyzwalacze

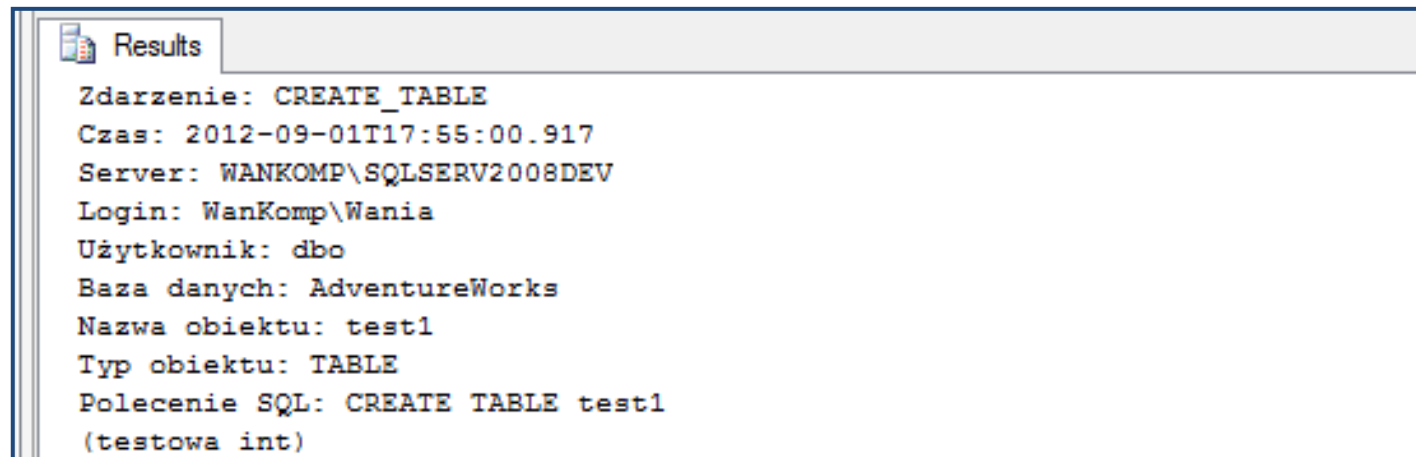
Wyzwalacze DDL – funkcja EVENTDATA

Strona 45

Testowanie wyzwalacza DDL z funkcją EVENTDATA:



Otrzymany komunikat:

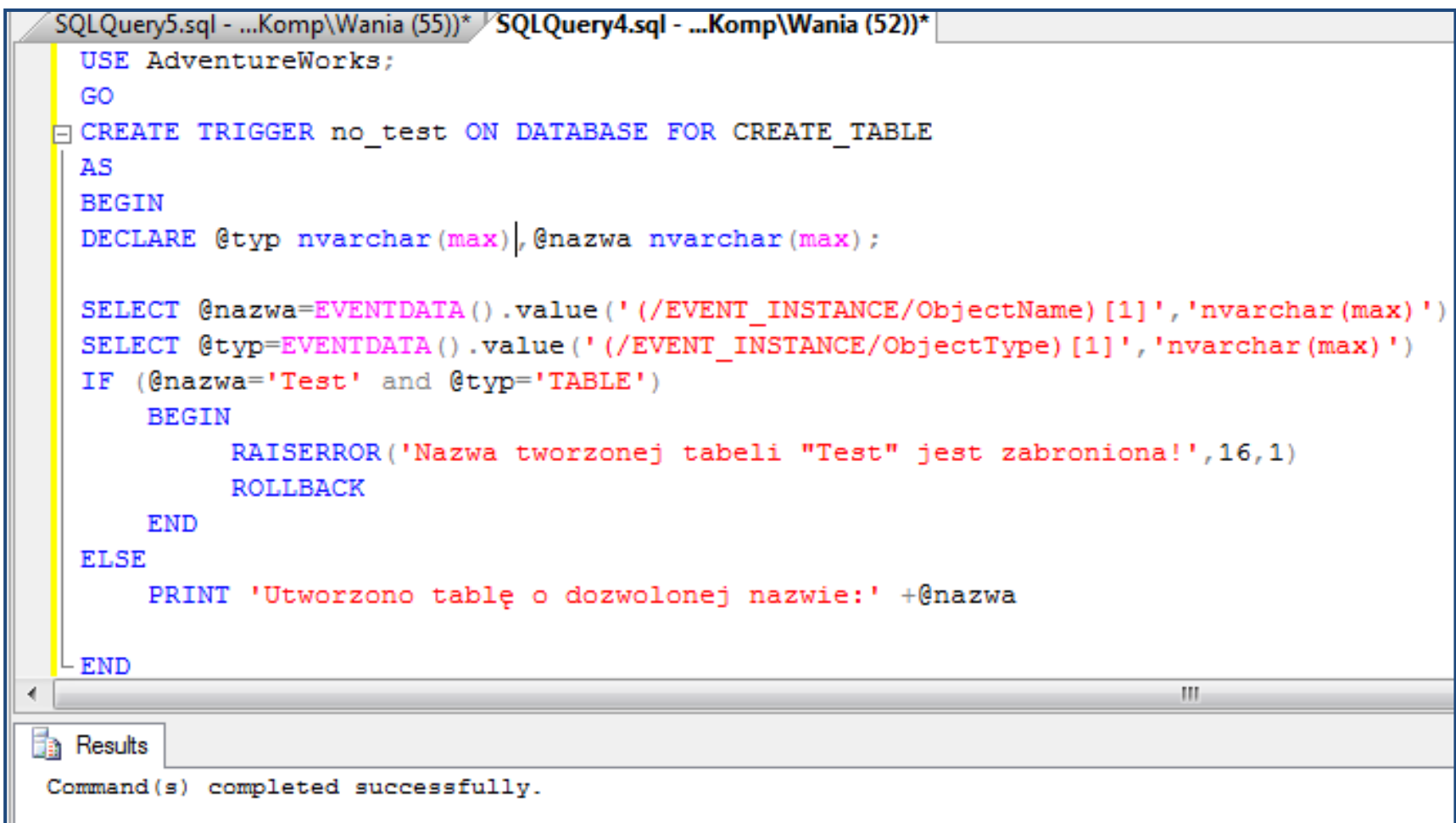


Wyzwalacze

Wyzwalacz DDL na poziomie bazy danych

Strona 46

Przykład tworzenia wyzwalacza anulującego tworzenie nowej tabeli o nazwie *Test* w bazie *AdventureWorks*:



```
SQLQuery5.sql - ...Komp\Wania (55))* SQLQuery4.sql - ...Komp\Wania (52))*
USE AdventureWorks;
GO
CREATE TRIGGER no_test ON DATABASE FOR CREATE_TABLE
AS
BEGIN
    DECLARE @typ nvarchar(max), @nazwa nvarchar(max);

    SELECT @nazwa=EVENTDATA().value('(/EVENT_INSTANCE/ObjectName) [1]', 'nvarchar(max)')
    SELECT @typ=EVENTDATA().value('(/EVENT_INSTANCE/ObjectType) [1]', 'nvarchar(max)')
    IF (@nazwa='Test' and @typ='TABLE')
    BEGIN
        RAISERROR('Nazwa tworzonej tabeli "Test" jest zabroniona!',16,1)
        ROLLBACK
    END
    ELSE
        PRINT 'Utworzono table o dozwolonej nazwie:' +@nazwa
END
```

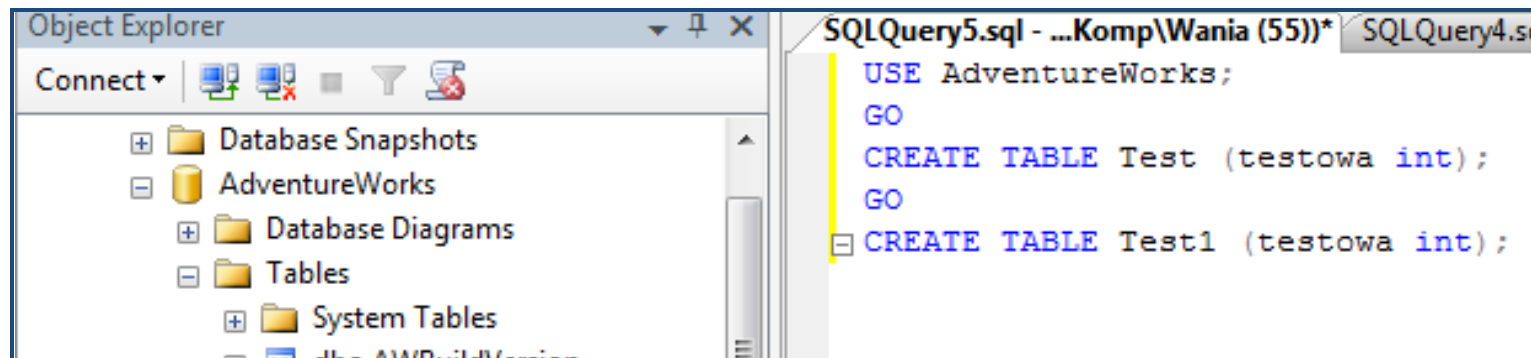
Results
Command(s) completed successfully.

Wyzwalacze

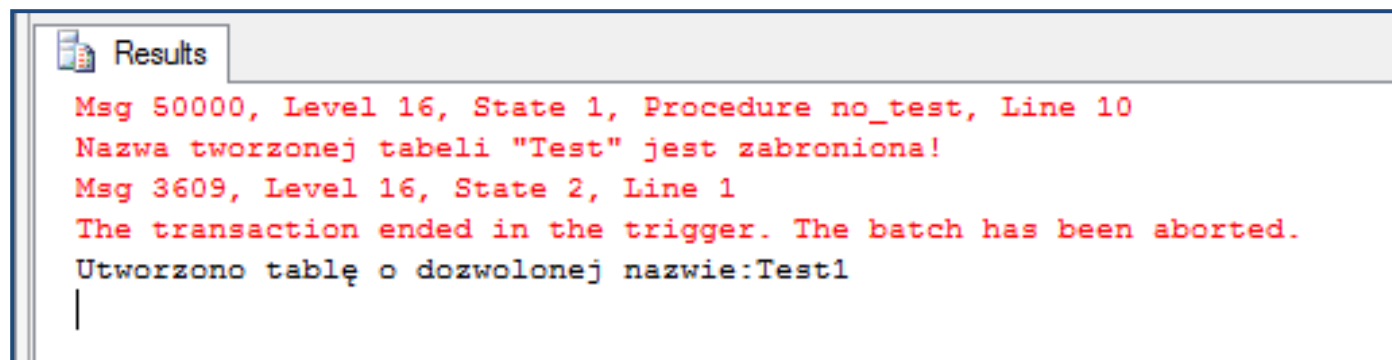
Wyzwalacz DDL na poziomie bazy danych

Strona 47

Testowanie wyzwalacza anulującego tworzenie nowej tabeli o nazwie *Test* w bazie *AdventureWorks*:



Zwrócone komunikaty:



Wyzwalacze

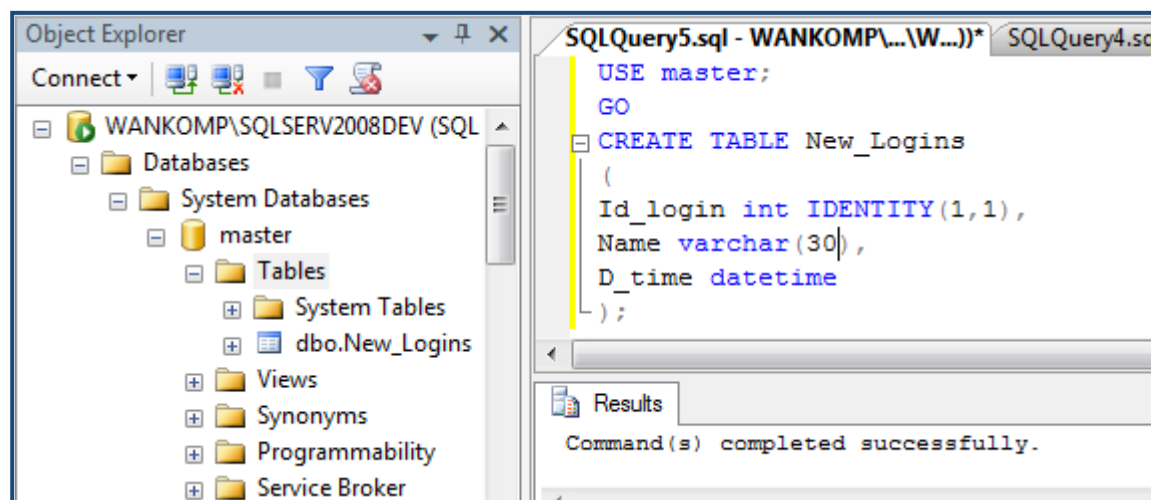
Wyzwalacz DDL na poziomie serwera

Strona 48

Wyzwalacze te mogą być definiowane dla zdarzeń zachodzących na poziomie serwera, np.: tworzenie, modyfikacja i usuwanie baz danych, loginów czy innych obiektów serwera.

Założono utworzenie wyzwalacza DDL na poziomie serwera, który będzie dodawał informację o nowo tworzonych loginach do dedykowanej tabeli w bazie *master*.

Przygotowanie tabeli do przechowywania danych o nowych loginach:

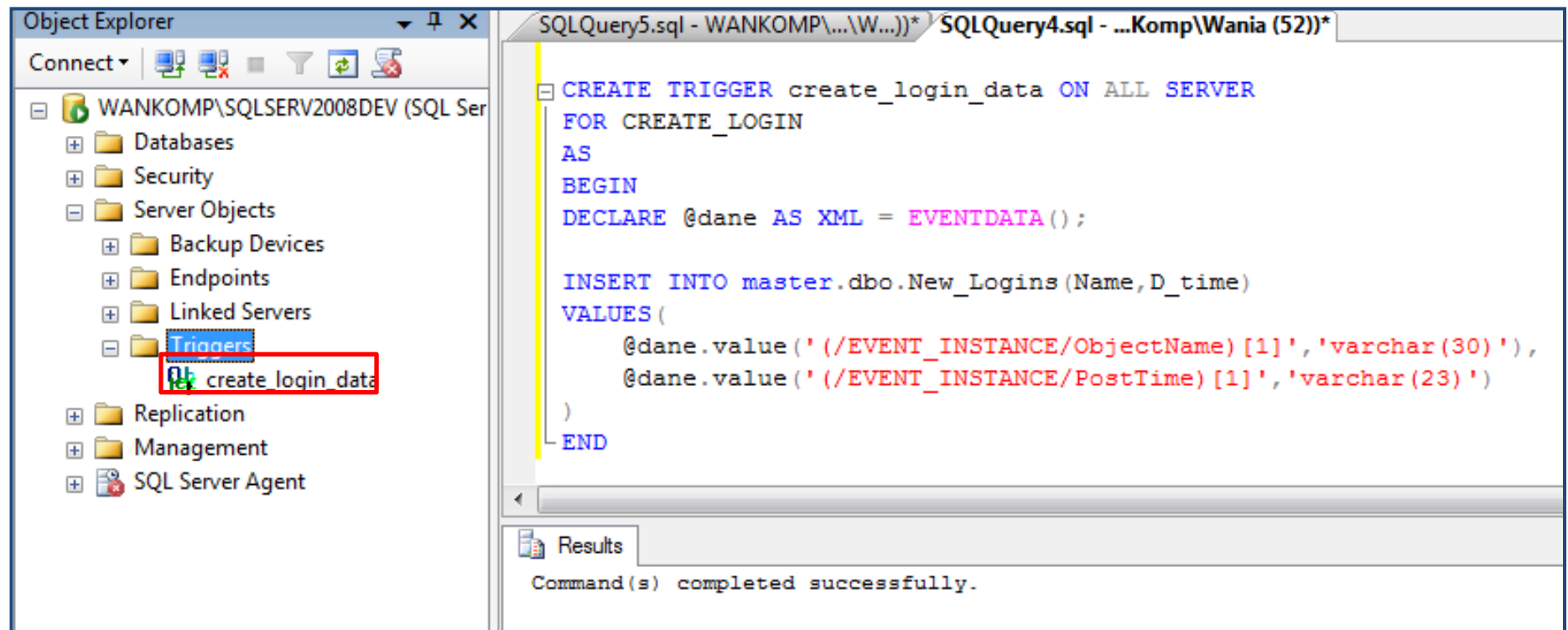


Wyzwalacze

Wyzwalacz DDL na poziomie serwera

Strona 49

Utworzenie zamierzonego wyzwalacza:



Wyzwalacze

Wyzwalacz DDL na poziomie serwera

Strona 50

Sprawdzenie działania utworzonego wyzwalacza:

The screenshot displays the SQL Server Enterprise Manager interface on the left and a SQL Query window on the right.

Object Explorer (Left): The tree view shows the server hierarchy for 'WANKOMP\SQLSERV2008DEV'. Under 'Security' > 'Logins', the logins 'login_testowy' and 'login_testowy2' are highlighted with a red rectangle.

SQL Query Window (Right): The query 'SQLQuery7.sql' contains the following T-SQL code:

```
USE master;
GO
CREATE LOGIN login_testowy WITH PASSWORD='testowe';
CREATE LOGIN login_testowy2 WITH PASSWORD='testowe2';
GO
SELECT * FROM New_Logins;
```

The 'Results' pane shows the output of the query, indicating that two rows were affected and displaying the details of the newly created logins:

Id_login	Name	D_time
3	login_testowy	2012-09-01 21:07:08.780
4	login_testowy2	2012-09-01 21:07:08.780

Wyzwalacze

Wyzwalacz DDL na poziomie serwera

Strona 51

Zdarzenia (DDL Events) które mogą uruchamiać wyzwalacze na poziomie serwera (źródło: <http://msdn.microsoft.com/en-us/library/bb522542.aspx>):

ALTER_AUTHORIZATION_SERVER	ALTER_SERVER_CONFIGURATION	ALTER_INSTANCE
CREATE_AVAILABILITY_GROUP	ALTER_AVAILABILITY_GROUP	DROP_AVAILABILITY_GROUP
CREATE_CREDENTIAL	ALTER_CREDENTIAL	DROP_CREDENTIAL
CREATE_CRYPTOGRAPHIC_PROVIDER	ALTER_CRYPTOGRAPHIC_PROVIDER	DROP_CRYPTOGRAPHIC_PROVIDER
CREATE_DATABASE	ALTER_DATABASE	DROP_DATABASE
CREATE_ENDPOINT	ALTER_ENDPOINT	DROP_ENDPOINT
CREATE_EVENT_SESSION	ALTER_EVENT_SESSION	DROP_EVENT_SESSION
CREATE_EXTENDED_PROCEDURE	DROP_EXTENDED_PROCEDURE	
CREATE_LINKED_SERVER	ALTER_LINKED_SERVER	DROP_LINKED_SERVER
CREATE_LINKED_SERVER_LOGIN	DROP_LINKED_SERVER_LOGIN	
CREATE_LOGIN	ALTER_LOGIN	DROP_LOGIN
CREATE_MESSAGE	ALTER_MESSAGE	DROP_MESSAGE
CREATE_REMOTE_SERVER	ALTER_REMOTE_SERVER	DROP_REMOTE_SERVER
CREATE_RESOURCE_POOL	ALTER_RESOURCE_POOL	DROP_RESOURCE_POOL
GRANT_SERVER	DENY_SERVER	REVOKE_SERVER
ADD_SERVER_ROLE_MEMBER	DROP_SERVER_ROLE_MEMBER	
CREATE_SERVER_AUDIT	ALTER_SERVER_AUDIT	DROP_SERVER_AUDIT
CREATE_SERVER_AUDIT_SPECIFICATION	ALTER_SERVER_AUDIT_SPECIFICATION	DROP_SERVER_AUDIT_SPECIFICATION
CREATE_WORKLOAD_GROUP	CREATE_WORKLOAD_GROUP	CREATE_WORKLOAD_GROUP

Wyzwalacze

Wyzwalacz DDL na poziomie bazy danych

Strona 52

Wybrane zdarzenia które mogą uruchamiać wyzwalacze na poziomie bazy danych (źródło:<http://msdn.microsoft.com/en-us/library /bb522542.aspx>):

...
CREATE_FULLTEXT_CATALOG	ALTER_FULLTEXT_CATALOG	DROP_FULLTEXT_CATALOG
CREATE_FULLTEXT_INDEX	ALTER_FULLTEXT_INDEX	DROP_FULLTEXT_INDEX
CREATE_FULLTEXT_STOPLIST	ALTER_FULLTEXT_STOPLIST	DROP_FULLTEXT_STOPLIST
CREATE_FUNCTION	ALTER_FUNCTION	DROP_FUNCTION
CREATE_INDEX	ALTER_INDEX	DROP_INDEX
CREATE_MASTER_KEY	ALTER_MASTER_KEY	DROP_MASTER_KEY
CREATE_MESSAGE_TYPE	ALTER_MESSAGE_TYPE	DROP_MESSAGE_TYPE
CREATE_PARTITION_FUNCTION	ALTER_PARTITION_FUNCTION	DROP_PARTITION_FUNCTION
CREATE_PARTITION_SCHEME	ALTER_PARTITION_SCHEME	DROP_PARTITION_SCHEME
CREATE_PLAN_GUIDE	ALTER_PLAN_GUIDE	DROP_PLAN_GUIDE
CREATE_PROCEDURE	ALTER_PROCEDURE	DROP_PROCEDURE
CREATE_QUEUE	ALTER_QUEUE	DROP_QUEUE
...
CREATE_STATISTICS	DROP_STATISTICS	UPDATE_STATISTICS
CREATE_SYMMETRIC_KEY	ALTER_SYMMETRIC_KEY	DROP_SYMMETRIC_KEY
CREATE_SYNONYM	DROP_SYNONYM	
CREATE_TABLE	ALTER_TABLE	DROP_TABLE
CREATE_TRIGGER	ALTER_TRIGGER	DROP_TRIGGER

- Wyzwalacze (trigery) umożliwiają automatyzację wielu procesów na serwerze bazy danych
- Dzięki temu zaoszczędzamy czas, pracę i redukujemy liczbę możliwych pomyłek (czynniki ludzkie)
- Mogą służyć do wymuszania złożonych reguł integralności danych, inspekcji dokonywanych zmian, bądź utrzymania pożądanego stanu danych.
- Do efektywnej implementacji wyzwalaczy warto wykorzystywać tabele specjalne INSERTED i DELETED
- W SQL Server możemy tworzyć wyzwalacze dla:
 - Zdarzeń DML
 - Zdarzeń DDL
 - Zdarzeń logowania

Transakcje, współbieżność i blokady

Wprowadzenie

Strona 54

Transakcję stanowi zbiór operacji, który zgodnie z regułami ACID traktowany jest jako:

- **Atomicity**(atomowy)- niepodzielny, wszystkie operacje są zrealizowane albo wycofane
- **Consistent** – na początku i końcu transakcji dane pozostają spójne
- **Isolation** – dane są odpowiednio wyizolowane(np. blokady), co uniemożliwia przeprowadzenie niezgodnych operacji
- **Durability** – trwałe, dane i ich zmiana zostaje na trwałe wprowadzone do bazy

W SQL Server transakcje rozpoczynamy poleceniem BEGIN TRAN, kończymy COMMIT TRAN, zmiany wycofujemy poleceniem ROLLBACK.

SQL Server nawet pojedynczą instrukcję bez jawnie zadeklarowanej transakcji traktuje jako transakcję.

Transakcje, współbieżność i blokady

Dziennik transakcji i pamięć podręczna.

Strona 55

- SQL Server dysponuje pamięcią podręczną, do której zapisywane są zmiany bazy danych.
- Jeżeli niezbędne dane nie znajdują się w pamięci podręcznej to najpierw są do niej pobierane z bazy danych i w pamięci podręcznej zapisywane są zmiany.
- Proces *checkpoint*, przenosi co jakiś czas zmodyfikowane strony z pamięci podręcznej do bazy danych, jednak tylko te dla których transakcje zostały już zapisane do dziennika transakcji.
- W przypadku awarii lub wycofania transakcji zmiany są wycofywane z bazy danych na podstawie wpisów w dzienniku transakcji.
- Informacje o transakcjach zapisane w dzienniku transakcji wykorzystywane są również w niektórych scenariuszach odtwarzania bazy danych.

Transakcje, współbieżność i blokady

Blokady

Strona 56

W bazach danych blokady odpowiadają za izolowanie transakcji, poprzez blokowanie różnego rodzaju dostępu do odpowiednich danych.

W SQL Server blokady możemy zakładać na zasoby na różnych poziomach szczegółowości:

- na poziomie wiersza
- na poziomie strony
- partycji
- tabele
- zakresy

W zapytaniach T-SQL możemy umieścić wskazówkę jaki szeroki zasób należy zablokować(wiersz, strona, tabela).

SQL Server nie musi uwzględnić tej wskazówki, np. w przypadku braku zasobów lub konfliktu z inną blokadą (np. blokadą wyłączną)

W SQL Server mamy dostępnych wiele różnych trybów blokad, które określają sposób uzyskiwania dostępu do blokowanych zasobów przez inne współbieżne transakcje.

Tryby blokad w SQL Server:

- Wyłączny(X) – jeśli inny proces będzie chciał założyć blokadę na zasoby zablokowane w tym trybie zostanie on zablokowany
- Dzielony (współdzielony)(S) – na ten sam zasób można nakładać wiele blokad dzielonych
- Modyfikacji(U) – pomaga zapobiegać zakleszczeniom, tylko jeden proces na dany zasób może założyć blokadę w tym trybie
- Intencyjny wyłączny(IX) – przed założeniem właściwej blokady żądane jest założenie blokady intencyjnej wyłącznej w celu wykrywania niezgodności blokad

Inne blokady w SQL Server:

- Blokada schematu – blokady na procesy operujące na schematach obiektu
 - Modyfikacji schematu – zakładana na obiekt dla instrukcji DDL
 - Stałości schematu – uniemożliwianie zmian schematu na obiekcie
- Blokada zbiorczej aktualizacji – uniemożliwienie zbiorczego(masowego) wstawiania danych do tabeli i blokowania wielu innych procesów
- Blokada zakresu klucza – służą do chronienia zakresów wierszy

SQL Server automatycznie określa tryb blokady, można jednak podać wskazówkę do której serwer może(ale nie musi) się zastosować.

Wskazówki można podać np. z wykorzystaniem opcji XLOCK, UPDLOCK:

```
SELECT .... FROM tabela (TABLELOCK)...
```

```
SELECT .... FROM tabela WITH(XLOCK, INDEX(index))
```

Transakcje, współbieżność i blokady

Zgodność blokad

Strona 59

W SQL Server mamy różne możliwości założenia blokady na zasoby, na które blokada już została założona w zależności od ich wcześniej omawianych (wraz z oznaczeniami) poszczególnych trybów:

Tryb żądany	Tryb założonej blokady					
	IS	S	U	IX	SIX	X
Intencyjny dzielony (IS)	Tak	Tak	Tak	tak	Tak	Nie
Dzielony (S)	Tak	Tak	Tak	Nie	Nie	Nie
Modyfikacji (U)	Tak	Tak	Nie	Nie	Nie	Nie
Intencyjny wyłączny (IX)	Tak	Nie	Nie	Tak	Nie	Nie
Dzielony z intencyjnym wyłącznym (SIX)	Tak	Nie	Nie	Nie	Nie	Nie
Wyłącznym (X)	Nie	Nie	Nie	Nie	Nie	Nie

Źródło: ms-melp://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_1devconc/html/452559b2-c536-43ec-a2da-a558abfa8a32.htm

Transakcje, współbieżność i blokady

Zablokowane transakcje

Strona 60

W wyniku nakładania przez serwer różnych blokad na różne zasoby podczas wykonywania różnych operacji, w związku z zgodnościami trybów blokad istnieje możliwość zablokowania wykonywanej transakcji.

W SQL Server mamy do dyspozycji szereg widoków oraz funkcji, które mogą się okazać znacząco pomocne w diagnozowaniu przyczyny zaistniałej sytuacji, np.:

- Widok ***sys.dm_tran_locks*** – informacje o blokadach
- Widok ***sys.dm_exec_connections*** – bieżące połączenia, w tym będące przyczyną istniejącej blokady
- Widok ***sys.dm_exec_sessions*** – informacje o sesjach
- Widok ***sys.dm_exec_requests*** – aktualnie wykonywane zapytania
- Funkcja ***sys.dm_exec_sql_text*** – funkcja do sprawdzania treści zapytań w wybranych sesjach

Transakcje, współbieżność i blokady

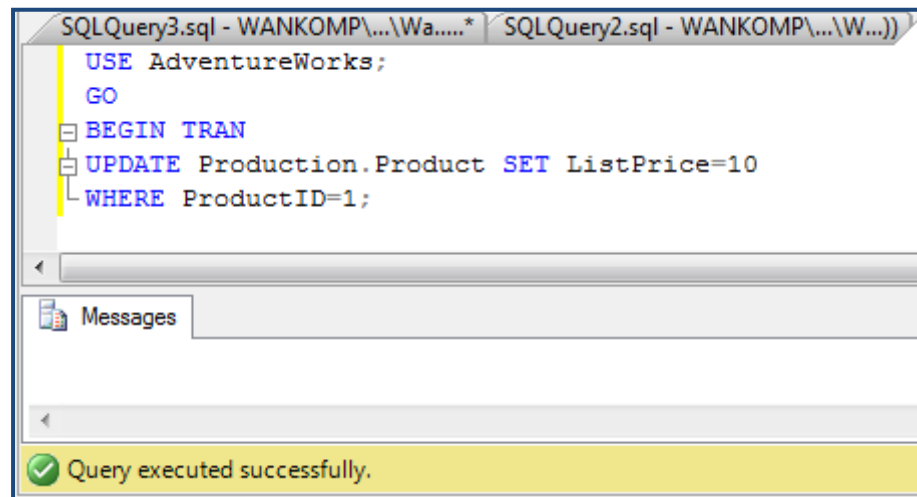
Modelowanie przykładowej blokady transakcji

Strona 61

Modelowanie blokady przykładowej transakcji zrealizowana na tabeli *Production.Product* testowej bazy danych *AdventureWorks*.

Transakcja nr1:

Polecenie UPDATE nakładające blokadę wyłączną wewnątrz transakcji:



Blokada wyłączna dla tego polecenia jest utrzymywana do końca transakcji.

Transakcja **nie została zakończona**, więc blokada jest cały czas **utrzymywana**.

Transakcje, współbieżność i blokady

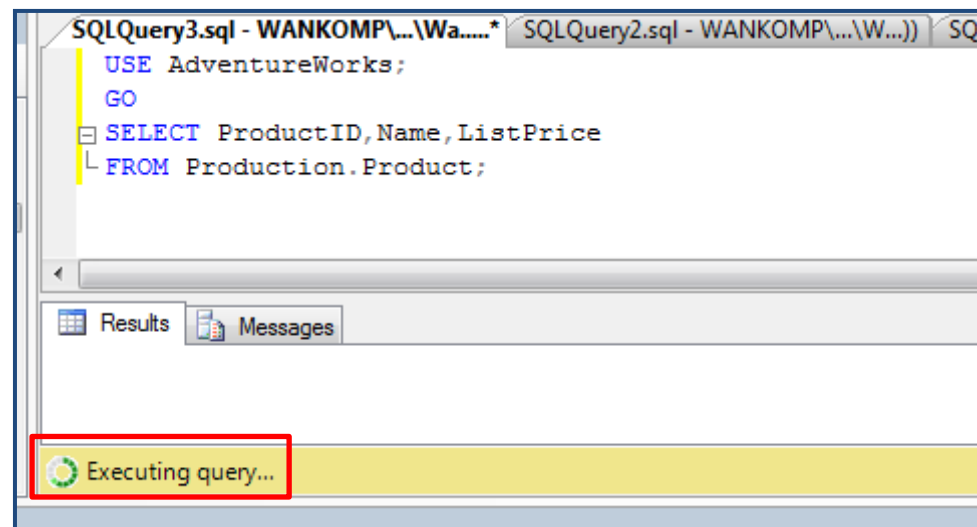
Modelowanie przykładowej blokady transakcji

Strona 62

W transakcji drugiej wykonujemy polecenie SELECT do tych samych zasobów, które wymaga dzielonej blokady do odczytu danych.

Transakcja nr2:

Polecenie SELECT równoległej transakcji:



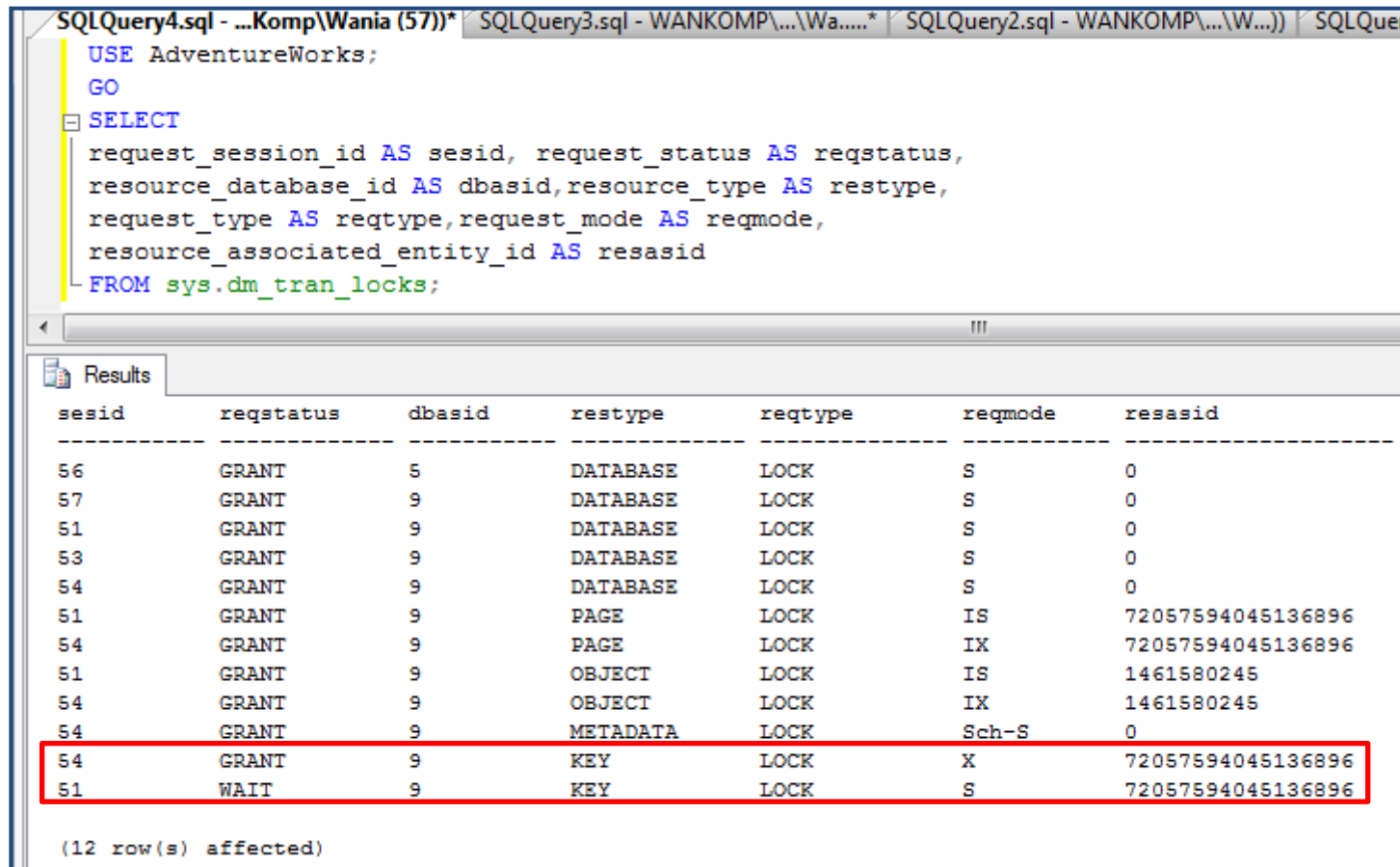
Zgodnie z przedstawioną tabelą zgodności blokad, utrzymywana blokada wyłączna uniemożliwia założenie blokady dzielonej na te same zasoby danych i transakcja nr2 zostaje zablokowana (Executing query...)

Transakcje, współbieżność i blokady

Diagnozowanie przyczyny blokady transakcji

Strona 63

W pierwszym etapie diagnozowania przyczyny zablokowanej transakcji możemy wykorzystać widok systemowy `sys.dm_tran_locks`



```
USE AdventureWorks;
GO
SELECT
    request_session_id AS sesid, request_status AS reqstatus,
    resource_database_id AS dbasid, resource_type AS restype,
    request_type AS reqtype, request_mode AS reqmode,
    resource_associated_entity_id AS resasid
FROM sys.dm_tran_locks;
```

sesid	reqstatus	dbasid	restype	reqtype	reqmode	resasid
56	GRANT	5	DATABASE	LOCK	S	0
57	GRANT	9	DATABASE	LOCK	S	0
51	GRANT	9	DATABASE	LOCK	S	0
53	GRANT	9	DATABASE	LOCK	S	0
54	GRANT	9	DATABASE	LOCK	S	0
51	GRANT	9	PAGE	LOCK	IS	72057594045136896
54	GRANT	9	PAGE	LOCK	IX	72057594045136896
51	GRANT	9	OBJECT	LOCK	IS	1461580245
54	GRANT	9	OBJECT	LOCK	IX	1461580245
54	GRANT	9	METADATA	LOCK	Sch-S	0
54	GRANT	9	KEY	LOCK	X	72057594045136896
51	WAIT	9	KEY	LOCK	S	72057594045136896

(12 row(s) affected)

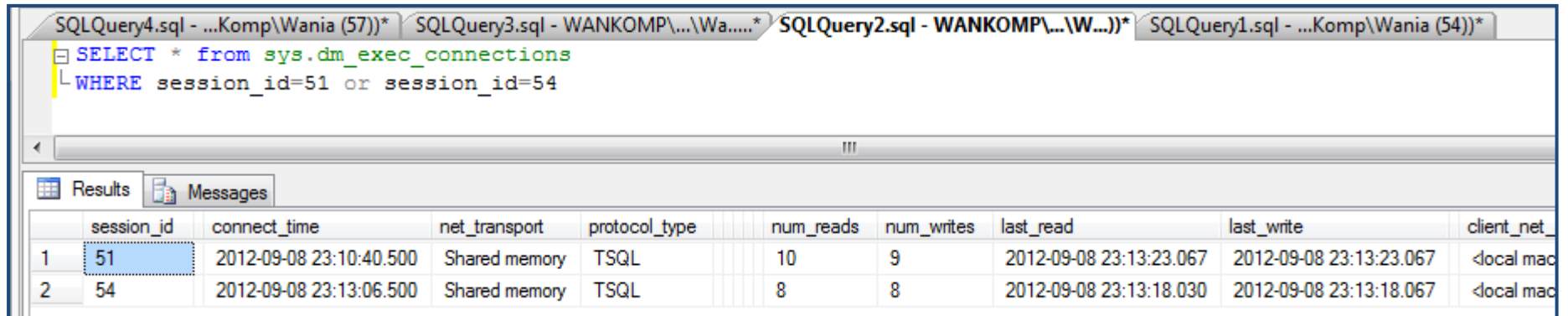
Proces 54 założył blokadę wyłączną, a proces 51 oczekuje na możliwość założenia blokady dzielonej na ten sam zasób.

Transakcje, współbieżność i blokady

Diagnozowanie przyczyny blokady transakcji

Strona 64

Poprzez widok `sys.dm_exec_connections` uzyskujemy informację o połączeniach dla tych procesów:



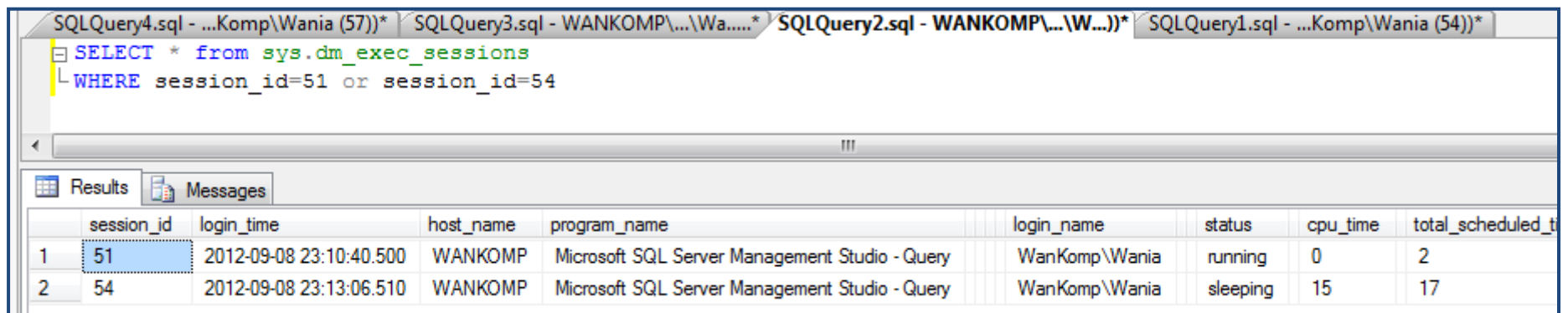
The screenshot shows a SQL query window with the following query:

```
SELECT * from sys.dm_exec_connections
WHERE session_id=51 or session_id=54
```

The results are displayed in a table with the following columns: session_id, connect_time, net_transport, protocol_type, num_reads, num_writes, last_read, last_write, and client_net_.

	session_id	connect_time	net_transport	protocol_type	num_reads	num_writes	last_read	last_write	client_net_
1	51	2012-09-08 23:10:40.500	Shared memory	TSQL	10	9	2012-09-08 23:13:23.067	2012-09-08 23:13:23.067	<local mac
2	54	2012-09-08 23:13:06.500	Shared memory	TSQL	8	8	2012-09-08 23:13:18.030	2012-09-08 23:13:18.067	<local mac

Również przydatne informacje możemy uzyskać z widoku `sys.dm_exec_sessions`



The screenshot shows a SQL query window with the following query:

```
SELECT * from sys.dm_exec_sessions
WHERE session_id=51 or session_id=54
```

The results are displayed in a table with the following columns: session_id, login_time, host_name, program_name, login_name, status, cpu_time, and total_scheduled_t.

	session_id	login_time	host_name	program_name	login_name	status	cpu_time	total_scheduled_t
1	51	2012-09-08 23:10:40.500	WANKOMP	Microsoft SQL Server Management Studio - Query	WanKomp\Wania	running	0	2
2	54	2012-09-08 23:13:06.510	WANKOMP	Microsoft SQL Server Management Studio - Query	WanKomp\Wania	sleeping	15	17

Transakcje, współbieżność i blokady

Diagnozowanie przyczyny blokady transakcji

Strona 65

Informację o aktualnie wykonywanych żądaniach uzyskujemy z widoku *sys.dm_exec_requests*:

SQLQuery4.sql - ...Komp\Wania (57))* SQLQuery3.sql - WANKOMP\...\Wa.....* SQLQuery2.sql - WANKOMP\...\W....)* SQLQuery1.sql - ...Komp\Wania (54))*

SELECT * from sys.dm_exec_requests

WHERE blocking_session_id>0;

Results

Messages

	session_id	request_id	start_time	status	command	sql_handle	statement_start_offset
1	51	0	2012-09-08 23:13:23.067	suspended	SELECT	0x02000000717ED40358F6164039941ACC3792436CA7942465	0

database_id	user_id	connection_id	blocking_session_id	wait_type	wait_time	last_wait_type	wait_resource
9	1	8DEAECDD-9D87-45BB-9269-03591A062513	54	LCK_M_S	4449617	LCK_M_S	KEY: 9:72057594045136896 (0100864707)

transaction_id	context_info	percent_complete	estimated_completion_time	cpu_time	total_elapsed_time	scheduler_id	task_address	reads	writes
1	26962	0x	0	0	4449617	2	0x000000000007234C8	0	0

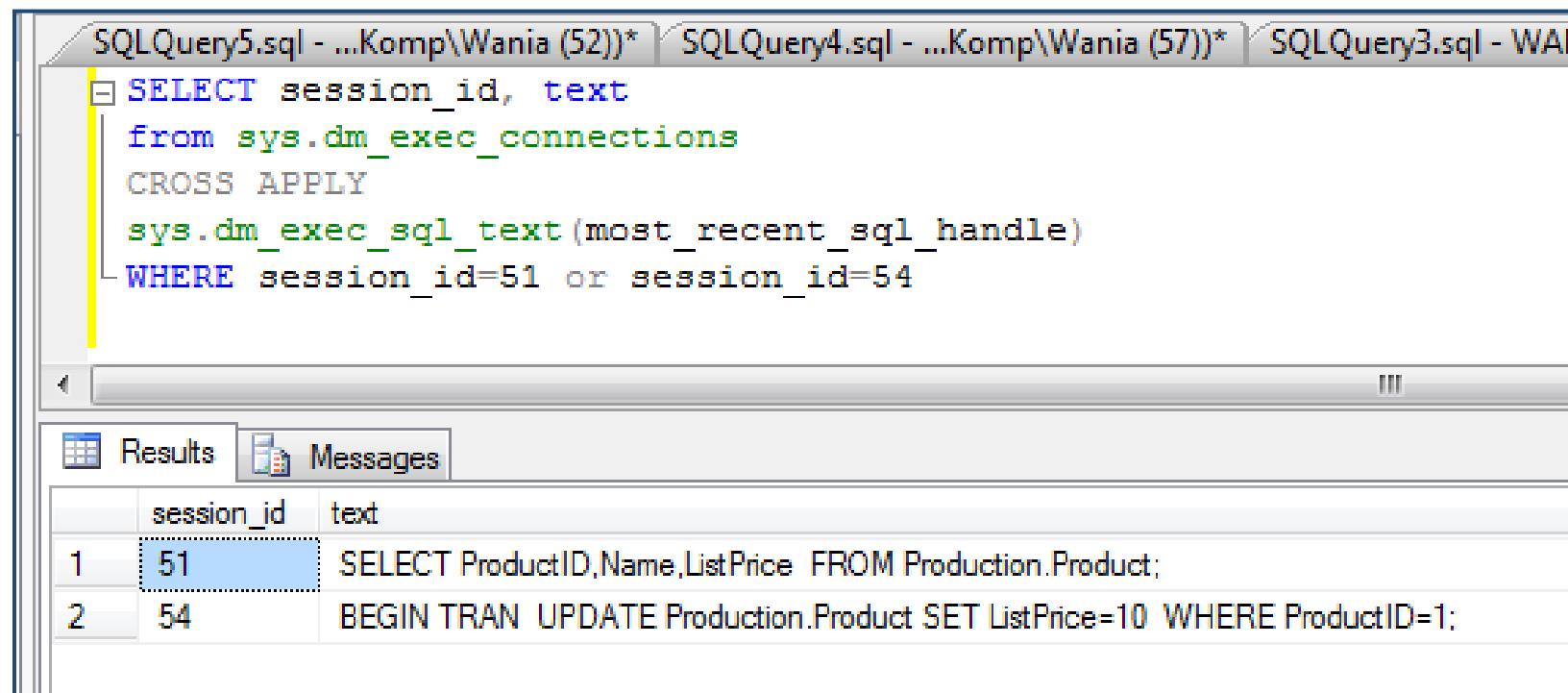
transaction_isolation_level	lock_timeout	deadlock_priority	row_count	prev_error	nest_level	granted_query_memory	executing_managed_code	group_id	query_text
1	2	-1	0	0	0	0	0	2	0xC

Transakcje, współbieżność i blokady

Diagnozowanie przyczyny blokady transakcji

Strona 66

Wykorzystując widok `sys.dm_exec_connections`, który zwraca uchwyt przekazany do funkcji `sys.dm_exec_sql_text` możemy sprawdzić treść ostatniego zapytania:



The screenshot shows a SQL Server Enterprise Manager interface. At the top, there are three tabs: 'SQLQuery5.sql - ...Komp\Wania (52))*', 'SQLQuery4.sql - ...Komp\Wania (57))*', and 'SQLQuery3.sql - WAI'. The active window displays a SQL query:

```
SELECT session_id, text
from sys.dm_exec_connections
CROSS APPLY
sys.dm_exec_sql_text(most_recent_sql_handle)
WHERE session_id=51 or session_id=54
```

Below the query window, there are two tabs: 'Results' and 'Messages'. The 'Results' tab is active, showing a table with two columns: 'session_id' and 'text'. The table contains two rows of data:

	session_id	text
1	51	SELECT ProductID,Name,ListPrice FROM Production.Product;
2	54	BEGIN TRAN UPDATE Production.Product SET ListPrice=10 WHERE ProductID=1;

W ten sposób uzyskaliśmy informację o poleceniach T-SQL, które są przyczyną powstałej blokady transakcji nr2.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji

Strona 67

W SQL Server mamy możliwość ustawienie 4 podstawowych poziomów izolacji transakcji (poziom domyślny READ COMMITTED):

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

, a także 2 dodatkowe:

- SNAPSHOT
- READ COMMITTED SNAPSHOT

Poziom izolacji na poziomie sesji ustawiamy następującym poleceniem

```
SET TRANSACTION ISOLATION LEVEL <....>
```

Transakcje, współbieżność i blokady

Zjawiska przy interakcji transakcji

Strona 68

Podczas równoległej realizacji wielu transakcji, zależnie od poziomu izolacji transakcji, mogą wystąpić, niektóre z niepożądanych zjawisk:

- BRUDNY ODCZYT – drugi proces odczytuje niezatwierdzone dane pierwszego procesu
- UTRACONE MODYFIKACJE – jeden proces może nadpisać modyfikacje wprowadzone przez inny proces
- ODCZYT NIE DAJĄCE SIĘ POWTÓRZYĆ – w tej samej transakcji w różnych odczytach tej samej danej uzyskujemy różne wartości
- FANTOMY – w tej samej transakcji w różnych odczytach z tymi samymi warunkami uzyskujemy różną liczbę wierszy wynikowych

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – READ UNCOMMITTED

Strona 69

Dla tego poziomu izolacji transakcji procesy nie wymagają zakładania blokad dzielonych do odczytu danych. Serwer przyjmuje założenie że spójność danych jest mniej istotna od jego wydajności.

Przykład niezatwierdzonego odczytu (brudnego odczytu):

The screenshot shows two SQL Server query windows. The left window, 'Transakcja1.sql', contains the following SQL code:

```
USE AdventureWorks;
GO
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
GO
BEGIN TRAN
UPDATE Production.Product SET ListPrice=10
WHERE ProductID=1;
```

The right window, 'Transakcja 2.sql', contains the following SQL code:

```
USE AdventureWorks;
GO
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
```

Below the code, the 'Results' pane for each transaction shows the data. Transaction 1's results show a ListPrice of 0,00. Transaction 2's results show a ListPrice of 10,00, indicating it read the uncommitted update from Transaction 1.

ProductID	Name	ListPrice
1	Adjustable Race	0,00

(1 row(s) affected)

ProductID	Name	ListPrice
1	Adjustable Race	10,00

(1 row(s) affected)

Po wykonaniu ROLBACK w pierwszej transakcji zmiana zostanie wycofana.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – REPEATABLE READ

Strona 70

Podobnie jak w READ COMMITTED zjawisko brudnego odczytu jest wyeliminowane, jednak transakcje utrzymują blokady dzielone aż do ich zakończenia, a nie tylko do zakończenia odczytu danych.

Przykład powtarzalnego odczytu (spójna analiza):

The screenshot shows two SQL Server query windows. The left window, titled 'Transakcja1.sql - WANKOMP\...\Wani...', contains the following SQL code:

```
USE AdventureWorks;
GO
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
```

The right window, titled 'Transakcja 2.sql - WANKOMP\...\Wan...', contains the following SQL code:

```
USE AdventureWorks;
GO
UPDATE Production.Product SET ListPrice=20
WHERE ProductID=1;
```

Below the code, the 'Results' pane for Transaction 1 shows a table with the following data:

ProductID	Name	ListPrice
1	Adjustable Race	10,00

The status bar at the bottom indicates that Transaction 1 has affected 1 row(s). Transaction 2 is currently 'Executing query...' and is blocked by Transaction 1.

Transakcja 2 zostaje zablokowana do momentu zakończenia transakcji 1.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – REPEATABLE READ

Strona 71

Przykład powtarzalnego odczytu (spójna analiza):

Transakcja 1.sql - WANKOMP\...\Wani...

```
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
```

ProductID	Name	ListPrice
1	Adjustable Race	10,00

(1 row(s) affected)

Transakcja 2.sql - WANKOMP\...\Wan...

```
UPDATE Production.Product SET ListPrice=20
WHERE ProductID=1;
```

RV2008DEV (10.... | WanKomp\Wania (51) | AdventureWorks | 00:00:00 | 1 rows | Executing query... | WANKOMP\

Poziom ten gwarantuje powtarzalny odczyt, gdyż inne procesy nie mogą założyć blokady wyłączonej pomiędzy odczytami, wyeliminowane są utracone modyfikacje (aktualizacje).

Transakcja 2 zostanie zakończona w momencie zakończenia transakcji 1 (COMMIT lub ROLLBACK)

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – SERIALIZABLE

Strona 72

Poziom ten gwarantuje powtarzalny odczyt oraz aktywne transakcje nakładają blokady zakresu klucza zgodnie z kryteriami zapytań dokonujących zarówno odczytu jak i zapisu. Jest to logiczne blokowanie danych spełniających kryteria. Na tym poziomie izolacji dodatkowo wyeliminowane jest zjawisko fantomów.

Przykład eliminacji odczytów fantomów:

The screenshot displays two SQL query windows side-by-side. The left window, titled 'Transakcja1.sql - WANKOMP\...\Wani...', contains the following SQL code:

```
USE AdventureWorks;
CREATE INDEX test_phantom ON
HumanResources.Department (GroupName);

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT DepartmentID, Name
FROM HumanResources.Department
WHERE GroupName='Manufacturing';
```

The 'Results' pane for this query shows two rows:

DepartmentID	Name
7	Production
8	Production Control

Below the results, it states '(2 row(s) affected)'. The right window, titled 'SQLQuery8.sql - WA...52)) Executing...*', contains the following SQL code:

```
INSERT INTO
HumanResources.Department (Name, GroupName, ModifiedDate)
VALUES ('Production 2', 'Manufacturing', GETDATE());
```

The 'Results' pane for this query is empty. The status bar at the bottom indicates that the first transaction is 'Executing query...' and the second transaction is 'WANKOMP\SQLSERV2008DEV (10.... WanKo'.

Ponownie transakcja 2 zostaje zablokowana.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – SERIALIZABLE

Strona 73

Przykład eliminacji odczytów fantomów:

W transakcji 1 ponownie uruchamiamy zapytanie SELECT:

The screenshot displays two SQL Server query windows side-by-side. The left window, titled 'Transakcja1.sql - WANKOMP\...\Wani...', contains a SELECT query: `SELECT DepartmentID, Name FROM HumanResources.Department WHERE GroupName='Manufacturing';`. Below the query, the 'Results' pane shows a table with two rows: (7, 'Production') and (8, 'Production Control'), with a message '(2 row(s) affected)'. The right window, titled 'SQLQuery8.sql - WA...52)) Executing...*', contains an INSERT query: `INSERT INTO HumanResources.Department (Name, GroupName, ModifiedDate) VALUES ('Production 2', 'Manufacturing', GETDATE());`. The 'Results' pane in this window is empty. The status bar at the bottom indicates 'Executing query...' and 'WANKOMP\SQLSERV2008DEV (10.... | WanKor'.

DepartmentID	Name
7	Production
8	Production Control

Widzimy, że dane z transakcji 2 nie zostały wprowadzone i w transakcji 1 mamy ponownie taki sam odczyt danych.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji bazujące na wersjach wierszy

Strona 74

SNAPSHOT – przypomina poziom izolacji transakcji SERIALIZABLE, jednak nie bazuje na mechanizmie blokowania, tylko na technologii przechowywania wersji wierszy. Operacje modyfikujące dane zapisują je w magazynie wierszy. Gdy transakcja modyfikująca dane jest otwarta, inna może zażądać starszej spójnej wersji wiersza. Ten poziom izolacji transakcji włączamy na poziomie całej bazy danych, np.:

```
ALTER DATABASE AdventureWorks SET ALLOW_SNAPSHOT_ISOLATION ON;
```

READ COMMITTED SNAPSHOT – bazująca na wersjach wierszy implementacja poziomu READ COMMITTED, również stosowana na poziomie bazy danych. W przeciwieństwie do poprzedniego poziomu nie implementuje funkcji wykrywania konfliktów oraz procesy uzyskują spójną wersję danych w momencie rozpoczęcia instrukcji, a nie jak poprzednio – transakcji. Szczególnie przydatny w aplikacjach przenoszonych z platform wspierających pobieranie starszych spójnych wersji danych.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji a niepożądane zjawiska

Strona 75

Ustawienie poziomu izolacji transakcji ma zasadniczy wpływ na możliwość występowanie niepożądanych zjawisk podczas ich interakcji oraz na szeroko pojętą wydajność systemu zarządzania bazą danych:

Tabela przedstawiająca możliwości występowania zjawisk w zależności od poziomu izolacji transakcji:

Poziom izolacji	Brudny odczyt	Odczyty nie dające się powtórzyć	Odczyty fantomy
Read uncommitted	Tak	Tak	Tak
Read committed	Nie	Tak	Tak
Repeatable read	Nie	Nie	Tak
Snapshot	Nie	Nie	Nie
Serializable	Nie	Nie	Nie

Źródło: ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_1devconc/html/8ac7780b-5147-420b-a539-4eb556e908a7.htm (SQL Server Books Online)

Transakcje, współbieżność i blokady

Zakleszczenia – charakterystyka ogólna

Strona 76

Wybrane cechy zjawiska zakleszczenia:

- Zakleszczenia powstają gdy dwa procesy blokują się wzajemnie.
- SQL Server automatycznie wykrywa zakleszczenia i rozwiązuje je poprzez przerwanie transakcji, która wykonała mniej czynności.
- Mamy możliwość określania dla sesji opcji wartości jej priorytetu w przypadku zakleszczenia (od -10 do 10), w tym przypadku ilość wykonanej pracy jest drugorzędna.
- Niekiedy zakleszczenia są pożądane, np. dla poziomu REPEATABLE READ zapobiegamy utraconym modyfikacjom poprzez powstawanie zakleszczeń.
- Chociaż powodują zachowanie spójności danych, w większości przypadków są niepożądane, gdyż powodują duże obciążenie.
- SQL Server dysponuje dedykowanymi narzędziami do śledzenia zdarzeń wpływających na zakleszczenia oraz informację bezpośrednio o zakleszczeniach.

Transakcje, współbieżność i blokady

Zakleszczenia – przykład

Strona 77

Prosty przykład zakleszczenia:

W dwóch różnych transakcjach dokonujemy modyfikacji wybranych wierszy w wybranych tabelach:

The screenshot displays two SQL query windows side-by-side, both connected to the same SQL Server instance (WANKOMP\SQLSERV2008DEV (10....)).

Left Window: Transakcja1.sql - WANKOMP\...\Wani...

```
USE AdventureWorks;
GO
BEGIN TRAN
UPDATE Production.Product SET ListPrice=ListPrice+10
WHERE ProductID=1;
```

Right Window: SQLQuery8.sql - ...Komp\Wania (52))*

```
USE AdventureWorks;
GO
BEGIN TRAN
UPDATE HumanResources.Department
SET GroupName='Manufacturing 2', ModifiedDate=GETDATE()
WHERE DepartmentID=1;
```

Both windows show the **Results** tab with the message: **(1 row(s) affected)**.

The status bar at the bottom indicates the query executed successfully: **Query executed succes... | WANKOMP\SQLSERV2008DEV (10.... | WanKomp\Wania**

Transakcje, współbieżność i blokady

Zakleszczenia – przykład

Strona 78

Prosty przykład zakleszczenia:

Następnie dokonujemy naprzemiennego odczytu modyfikowanych danych w równoległych transakcjach oraz zatwierdzamy transakcję:

The screenshot displays two SQL query windows side-by-side. The left window, titled 'Transakcja1.sql - WANKOMP\...\Wani...', contains the following SQL code:

```
SELECT GroupName
FROM HumanResources.Department
WHERE DepartmentID=1;
COMMIT;
```

Below the code, the 'Results' pane shows a single row with the value 'Manufacturing 2' under the column 'GroupName'. The status bar at the bottom of this window indicates '(1 row(s) affected)'.

The right window, titled 'SQLQuery8.sql - ...Komp\Wania (52))*', contains the following SQL code:

```
SELECT ListPrice
FROM Production.Product
WHERE ProductID=1;
COMMIT;
```

Below the code, the 'Results' pane shows a single row with the value 'ListPrice' under the column 'ListPrice'. The status bar at the bottom of this window indicates 'Query completed with...'.

The bottom status bar of the entire interface shows the following information: 'SQLSERV2008DEV (10.... | WanKomp\Wania (51) | AdventureWorks | 00:00:16 | 1 rows'. A yellow warning icon is visible next to the status bar, indicating an error.

SQL Server automatycznie przerywa jedną z transakcji po wykryciu zakleszczenia. Gdyby przetwarzać tabele **w tej samej kolejności** można uniknąć tego zakleszczenia.

Transakcje, współbieżność i blokady

Podsumowanie

Strona 79

- Podczas przetwarzania informacji równolegle dla wielu użytkowników warto pamiętać o możliwości wykorzystywania transakcji.
- Projektując aplikację dla wielu użytkowników należy rozważyć różne modele współbieżności.
- Różne poziomy izolacji transakcji ograniczają występowanie różnych niepożądanych zjawisk podczas ich interakcji, co nie pozostaje bez znaczenia na szeroko pojętą wydajność serwera bazy danych.
- SQL Server wspiera poziomy izolacji transakcji bazujące na blokowaniu oraz na wersjach wierszy na poziomie całej bazy danych.
- Podczas tworzenia kodu T_SQL należy wziąć pod uwagę niektóre zalecenia, np. przetwarzanie tabel w tej samej kolejności w różnych transakcjach, w celu uniknięcia jak największego zbioru scenariuszy powstawania zakleszczeń.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – READ COMMITTED

Strona 80

Poziom ten jest poziomem domyślnym w SQL Server i eliminuje zjawisko brudnego odczytu. Procesy żądają nałożenie blokady dzielonej podczas odczytu danych.

Test niezatwierdzonego odczytu (brudnego odczytu):

The screenshot displays two SQL query windows side-by-side. The left window, titled 'Transakcja1.sql', contains the following SQL code:

```
USE AdventureWorks;
GO
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
GO
BEGIN TRAN
UPDATE Production.Product SET ListPrice=10
WHERE ProductID=1;
```

The right window, titled 'Transakcja 2.sql', contains the following SQL code:

```
USE AdventureWorks;
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
```

Below the query windows, the 'Results' pane for Transaction 1 shows a table with the following data:

ProductID	Name	ListPrice
1	Adjustable Race	0,00

Below the table, it indicates '(1 row(s) affected)' and '(1 row(s) affected)'. The status bar at the bottom shows 'RV2008DEV (10.... | WanKomp\Wania (51) | AdventureWorks | 00:00:00 | 1 rows' and 'Executing query... | WANKOMP\SQLSER'.

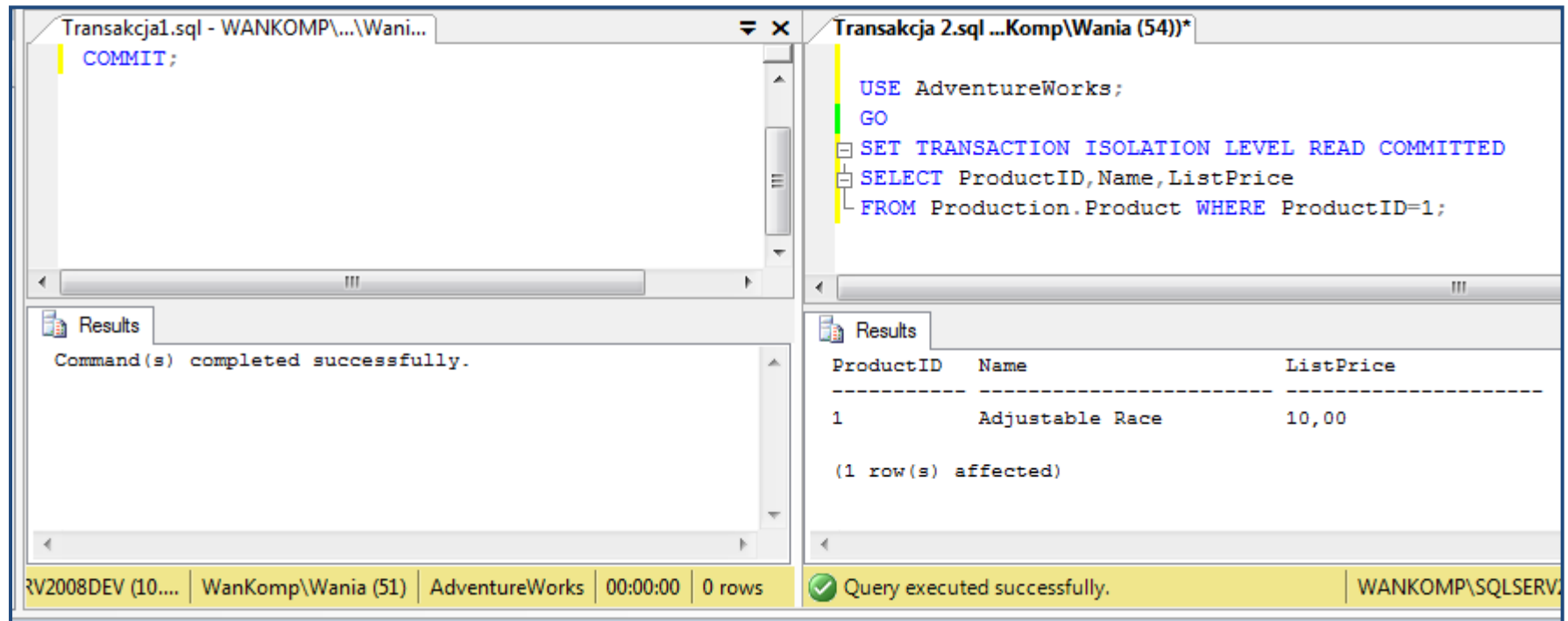
Proces transakcji 2 zostaje zablokowany do momentu zakończenia transakcji 1.

Transakcje, współbieżność i blokady

Poziomy izolacji transakcji – READ COMMITTED

Strona 81

Test niezatwierdzonego odczytu (brudnego odczytu):



The screenshot displays two SQL query windows side-by-side. The left window, titled 'Transakcja1.sql - WANKOMP\...\Wani...', contains the command `COMMIT;`. The right window, titled 'Transakcja 2.sql ...Komp\Wania (54))', contains the following SQL code:

```
USE AdventureWorks;
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT ProductID, Name, ListPrice
FROM Production.Product WHERE ProductID=1;
```

Below the code in the right window, the 'Results' pane shows the output of the query:

ProductID	Name	ListPrice
1	Adjustable Race	10,00

Below the table, it indicates '(1 row(s) affected)'. At the bottom of the interface, a status bar shows 'Query executed successfully.' and 'WANKOMP\SQLSERV...'.

Po zatwierdzeniu transakcji 1 proces transakcji 2 zostaje odblokowany i następuje odczyt zmienionych i zatwierdzonych danych.

Poziom READ COMMITTED nie eliminuje jednak występowania innych problemów związanych ze współbieżnością transakcji.

W przypadku gdy wykonanie kodu Transact SQL nie zostanie poprawnie zrealizowane w SQL Server mamy możliwość obsłużenia błędów (ograniczone ale zawsze;)).

W SQL Server 2008 mamy dwie możliwości obsługi błędów:

- Z wykorzystaniem funkcji `@@ERROR`
- Z wykorzystaniem bloku `TRY/CATCH` (od SQL Server 2005)

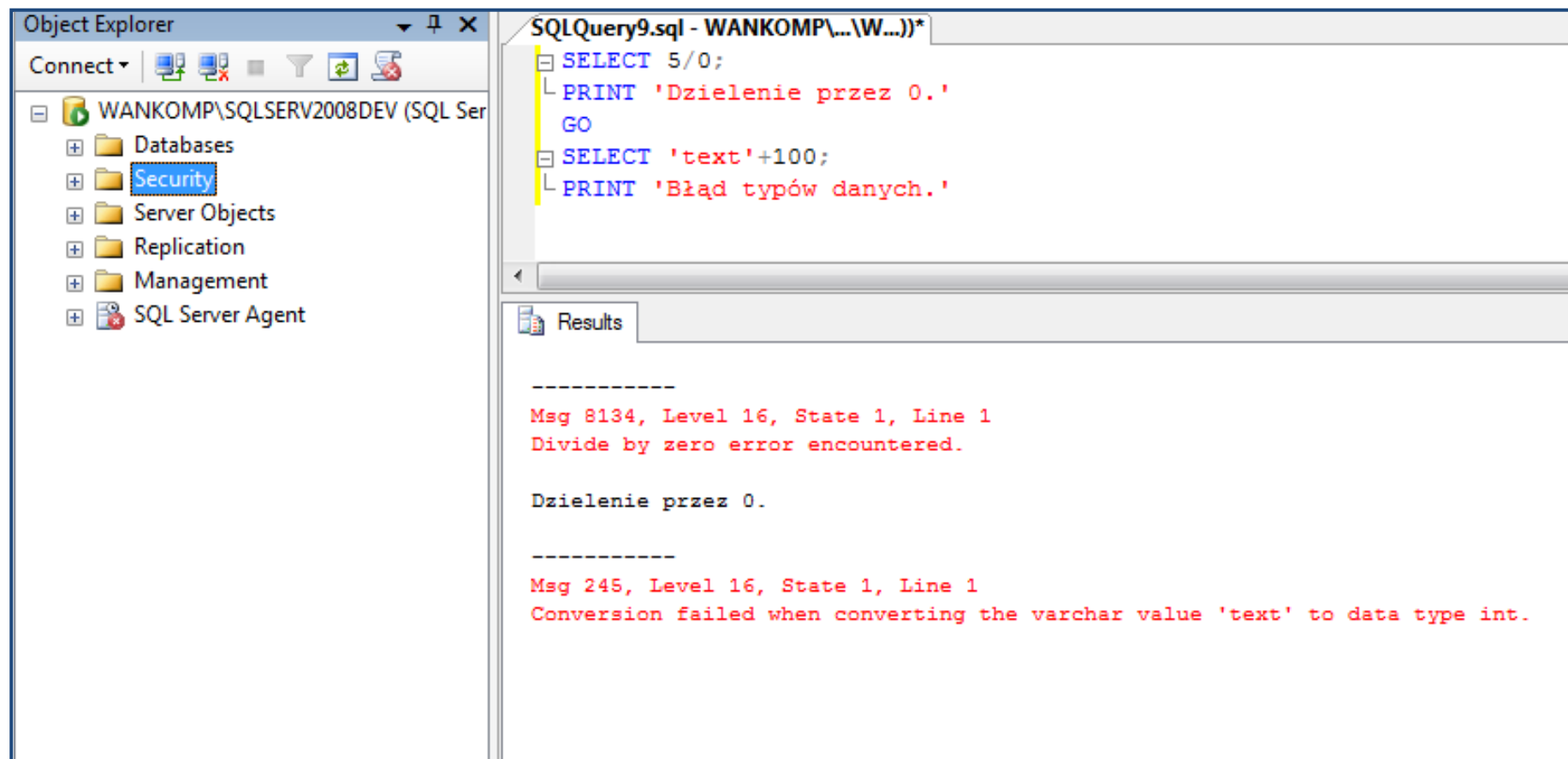
UWAGA:

Niektóre błędy prowadzą po prostu do przerwania wykonywania kodu i nie możemy ich przechwycić (jedynie rozwiązanie po stronie systemu). Błędy które powodują przerwanie wsadu to np.: błędy konwersji i zakleszczenia.

Obsługa błędów

Błędy powodujące przerwanie wykonywania kodu T-SQL

Strona 83



Uwaga: Błąd konwersji typów danych powoduje przerwanie kodu (tekst „Błąd typów danych.” nie został wyświetlony), w przeciwieństwie do błędu dzielenia przez wartość 0.

Obsługa błędów

Z wykorzystaniem @@ERROR oraz @@ROWCOUNT

Strona 84

Przykład wykorzystania funkcji @@ERROR i @@ROWCOUNT dla operacji dzielenia:

```
SQLQuery9.sql - WANKOMP\...\W...)*
SELECT 5/1;
SELECT @blad=@@ERROR,@wiersze=@@ROWCOUNT
IF (@blad=0 and @wiersze>0)
PRINT 'Operacja została wykonana poprawnie dla ' + CAST(@wiersze AS varchar(10))+' wierszy.'
ELSE
PRINT 'Wystąpił błąd o numerze:' + CAST(@blad AS varchar(10))

SELECT 5/0;
SELECT @blad=@@ERROR,@wiersze=@@ROWCOUNT
IF (@blad=0 and @wiersze>0)
PRINT 'Operacja została wykonana poprawnie dla ' + CAST(@wiersze AS varchar(10))+' wierszy.'
ELSE
PRINT 'Wystąpił błąd o numerze:' + CAST(@blad AS varchar(10))
```

Results

(1 row(s) affected)

Operacja została wykonana poprawnie dla 1 wierszy.

Msg 8134, Level 16, State 1, Line 9
Divide by zero error encountered.

Wystąpił błąd o numerze:8134

Obsługa błędów

Z wykorzystaniem TRY/CATCH

Strona 85

Składnia polecenia TRY/CATCH w języku T-SQL:

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

W bloku TRY umieszczamy wsad T-SQL, który może generować błędy, natomiast w bloku CATCH umieszczamy wsad będący reakcją na pojawienie się błędu.

SQL Server udostępnia dodatkowe funkcje obsługi błędów jak `ERROR_NUMBER()`, `ERROR_MESSAGE()`, `ERROR_LINE()` ...

Obsługa błędów

Z wykorzystaniem TRY/CATCH

Strona 86

Przykład wykorzystania polecenia TRY/CATCH oraz funkcji informujących o zaistniałym błędzie:

```
SQLQuery10.sql - WANKOMP\...\W...)* SQLQuery9.sql - WANKOMP\...\W...)*
BEGIN TRY
    -- blok podejrzanych instrukcji mogących powodować powstanie błędu
    DECLARE @liczba int;
    -- instrukcja poprawna
    SELECT @liczba=5/1;PRINT 'Wynik dzielenia='+ CAST(@liczba AS nvarchar(10));
    --instrukcja powodująca błąd dzielenia przez 0
    SELECT @liczba=5/0;PRINT 'Wynik dzielenia='+ CAST(@liczba AS nvarchar(10));
    --instrukcja poprawna
    SELECT @liczba=5/5;PRINT 'Wynik dzielenia='+ CAST(@liczba AS nvarchar(10));
END TRY
BEGIN CATCH
    --blok wsadu kodu będącego reakcją na powstanie błędu
    DECLARE @i AS int, @text AS nvarchar(max);

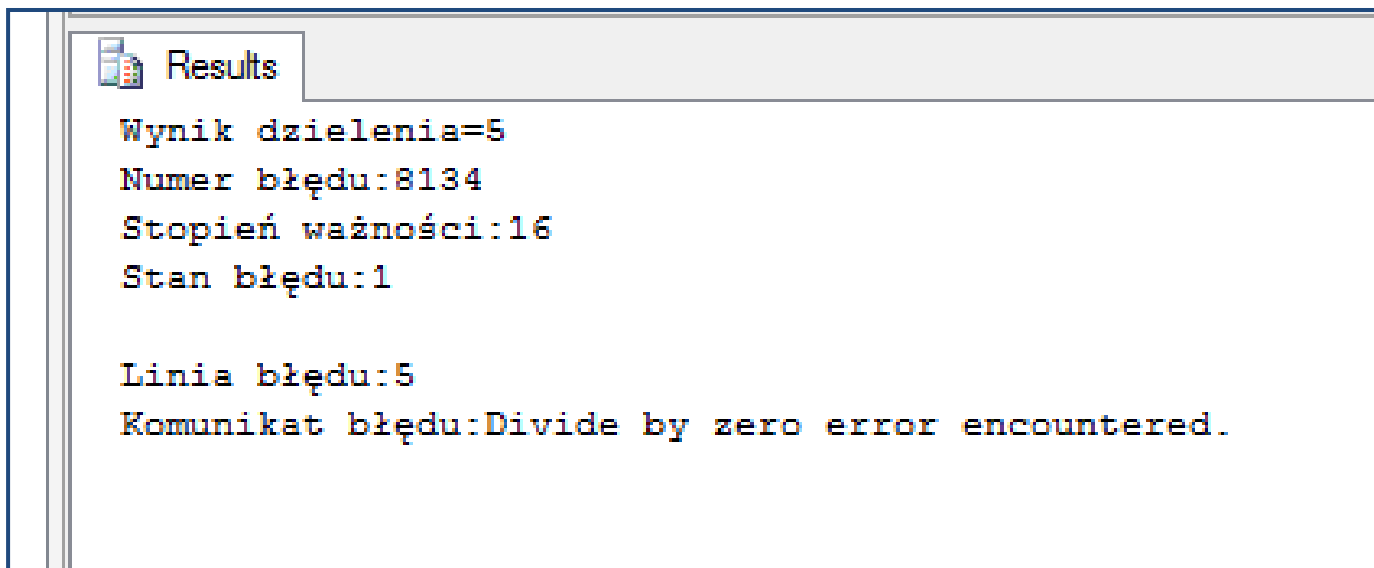
    -- wyświetlenie informacji o błędzie z wykorzystaniem wbudowanych funkcji
    SELECT @i=ERROR_NUMBER(); PRINT 'Numer błędu:'+CAST(@i as nvarchar(10));
    SELECT @i=ERROR_SEVERITY(); PRINT 'Stopień ważności:'+CAST(@i as nvarchar(10));
    SELECT @i=ERROR_STATE(); PRINT 'Stan błędu:'+CAST(@i as nvarchar(10));
    SELECT @text=ERROR_PROCEDURE(); PRINT 'Procedura:'+@text;
    SELECT @i=ERROR_LINE(); PRINT 'Linia błędu:'+CAST(@i as nvarchar(10));
    SELECT @text=ERROR_MESSAGE(); PRINT 'Komunikat błędu:'+@text;
END CATCH;
```

Obsługa błędów

Z wykorzystaniem TRY/CATCH

Strona 87

Sprawdzenie wykonania polecenia TRY/CATCH oraz funkcji informujących o zaistniałym błędzie:



```
Results
Wynik dzielenia=5
Numer błędu:8134
Stopień ważności:16
Stan błędu:1

Linia błędu:5
Komunikat błędu:Divide by zero error encountered.
```

Uwaga:

Warto zauważyć, że poprawna instrukcja, występująca przed instrukcją generującą błąd została wykonana. Natomiast poprawna instrukcja umieszczona za instrukcją generującą błąd nie została już wykonana.

Obsługa błędów

Z wykorzystaniem TRY/CATCH – funkcja XACT_STATE

Strona 88

W przypadku wystąpienia błędu funkcja XACT_STATE zwraca wartość zależną od stanu przerwanej transakcji.

Przykład wykorzystania funkcji XACT_STATE:

```
BEGIN TRY
    BEGIN TRAN
        -- operacje transakcji
    COMMIT TRAN
    PRINT 'Operacje zrealizowano poprawnie.'
END TRY
BEGIN CATCH
    IF (XACT_STATE())=-1
        --Transakcja otwarta - nie można zatwierdzić.
        ROLLBACK
    ELSE IF (XACT_STATE())=1
        --Transakcja otwarta - można zatwierdzić.
        COMMIT
    ELSE
        PRINT ' Nie ma otwartych transakcji'
END CATCH;
```


- Wprowadzenie konstrukcji TRY/CATCH stanowiło ogromny przełom w dziedzinie obsługi błędów w SQL Server
- Dzięki TRY/CATCH kod obsługi błędów jest uporządkowany i czytelny
- Dzięki TRY/CATCH możemy obsłużyć błędy z wyjątkiem tych krytycznych, np. błędu konwersji danych
- W bloku CATCH możemy wykorzystywać dodatkowe funkcję dostarczające informacje o błędach
- Przy stosowaniu bloku TRY/CATCH i pojawieniu się błędu w jawnej transakcji to sesja może zakończyć się stanem transakcji:
 - Brak otwartych transakcji
 - Transakcja otwarta i możliwa do zatwierdzenia
 - Transakcja otwarta i niemożliwa do zatwierdzenia

SQL Server charakteryzuje się szerokim wsparciem dla obsługi danych typu XML:

- Tworzenie dokumentów XML na podstawie danych relacyjnych
- Transformacja dokumentów XML do postaci relacyjnej
- Wbudowany typ danych XML
- Ograniczanie danych XML przy użyciu schematów
- Tworzenie indeksów XML
- T-SQL akceptuje ciągi XQuery jako parametry

Do generowania dokumentów XML na podstawie danych relacyjnych służy polecenie `SELECT ... FOR XML`

SQL Server oferuje następujące warianty tego polecenia:

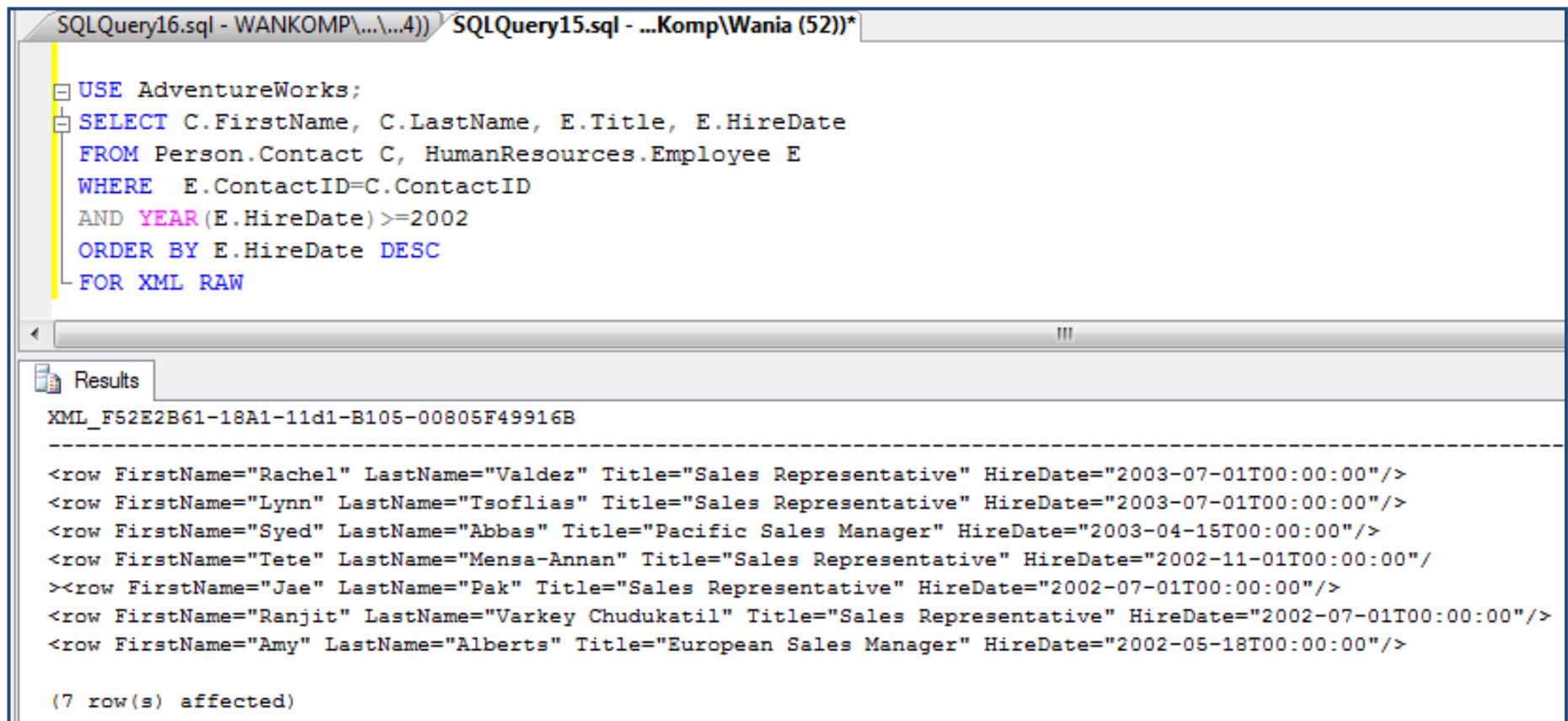
- `...FOR XML RAW` – przypomina tabelaryczną postać (wiersze do atrybutu *row*, kolumny konwertowane do atrybutów)
- `...FOR XML AUTO` – tworzenie dokumentów XML skoncentrowanych na elementach
- `...FOR XML EXPLICIT` – własnoręczne definiowanie dokumentów XML z wykorzystaniem przestarzałego T-SQL
- `...FOR XML PATH` – własnoręczne definiowanie dokumentów XML z wykorzystaniem XPATH

Dane XML

Generowanie dokumentów XML – FOR XML RAW

Strona 92

Przykład wygenerowania dokumentu XML z wybranymi danymi pracowników testowej bazy *AdventureWorks*:



The screenshot shows a SQL Server window with two tabs: 'SQLQuery16.sql - WANKOMP\...\...4))' and 'SQLQuery15.sql - ...Komp\Wania (52))*'. The active tab displays a SQL query that selects employee data from the AdventureWorks database, ordered by hire date in descending order, and formats the output as XML using the FOR XML RAW clause. Below the query, the 'Results' pane shows the generated XML document, which contains seven rows of employee data, each represented as an XML element with attributes for first name, last name, title, and hire date. The results pane also indicates that 7 rows were affected.

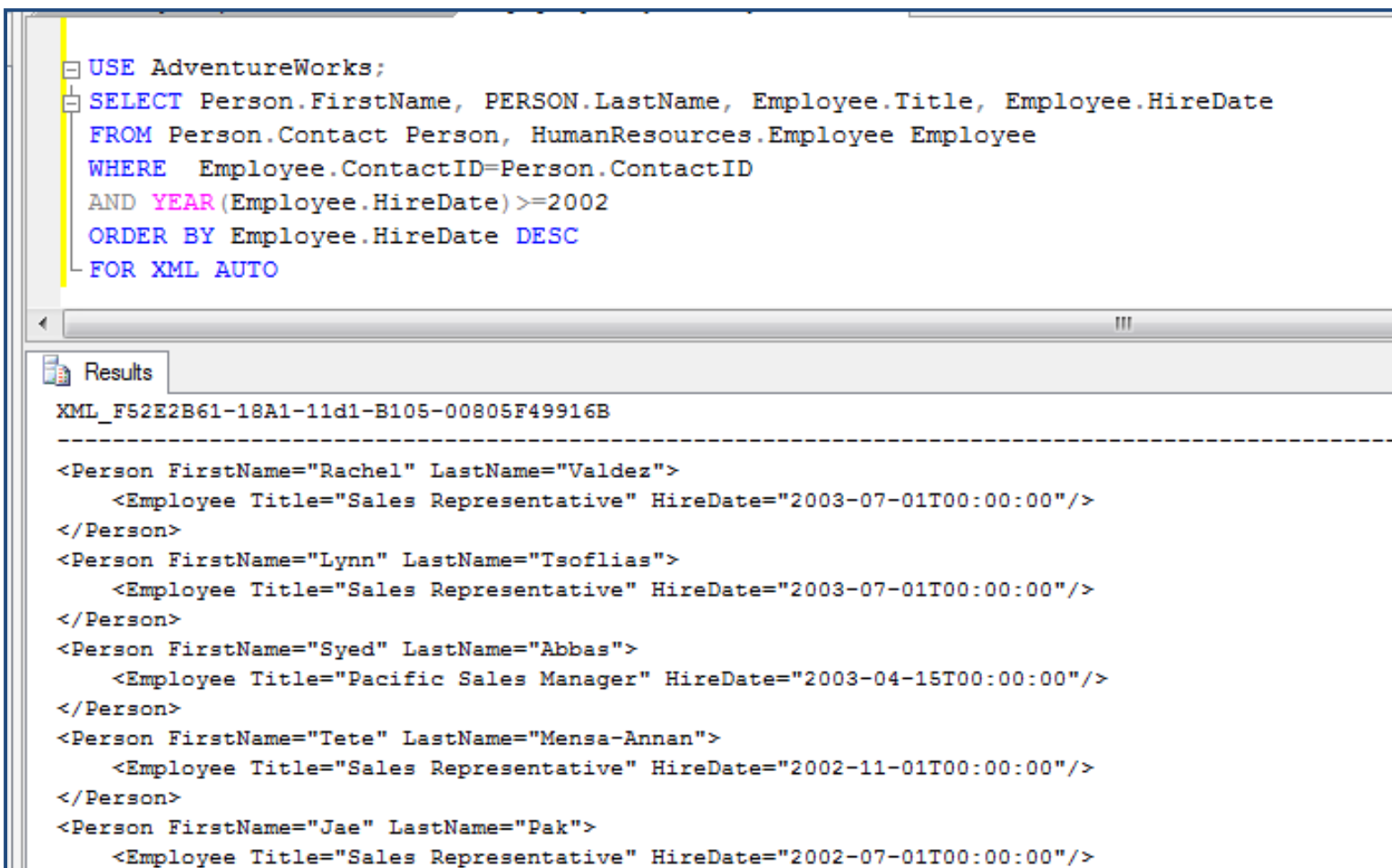
```
SQLQuery16.sql - WANKOMP\...\...4)) SQLQuery15.sql - ...Komp\Wania (52))*  
  
USE AdventureWorks;  
SELECT C.FirstName, C.LastName, E.Title, E.HireDate  
FROM Person.Contact C, HumanResources.Employee E  
WHERE E.ContactID=C.ContactID  
AND YEAR(E.HireDate)>=2002  
ORDER BY E.HireDate DESC  
FOR XML RAW  
  
Results  
XML_F52E2B61-18A1-11d1-B105-00805F49916B  
-----  
<row FirstName="Rachel" LastName="Valdez" Title="Sales Representative" HireDate="2003-07-01T00:00:00"/>  
<row FirstName="Lynn" LastName="Tsoflias" Title="Sales Representative" HireDate="2003-07-01T00:00:00"/>  
<row FirstName="Syed" LastName="Abbas" Title="Pacific Sales Manager" HireDate="2003-04-15T00:00:00"/>  
<row FirstName="Tete" LastName="Mensa-Annan" Title="Sales Representative" HireDate="2002-11-01T00:00:00"/>  
><row FirstName="Jae" LastName="Pak" Title="Sales Representative" HireDate="2002-07-01T00:00:00"/>  
<row FirstName="Ranjit" LastName="Varkey Chudukatil" Title="Sales Representative" HireDate="2002-07-01T00:00:00"/>  
<row FirstName="Amy" LastName="Alberts" Title="European Sales Manager" HireDate="2002-05-18T00:00:00"/>  
  
(7 row(s) affected)
```

Dane XML

Generowanie dokumentów XML – FOR XML AUTO

Strona 93

Przykład wygenerowania dokumentu XML z wybranymi danymi pracowników testowej bazy *AdventureWorks*:



```
USE AdventureWorks;
SELECT Person.FirstName, PERSON.LastName, Employee.Title, Employee.HireDate
FROM Person.Contact Person, HumanResources.Employee Employee
WHERE Employee.ContactID=Person.ContactID
AND YEAR(Employee.HireDate)>=2002
ORDER BY Employee.HireDate DESC
FOR XML AUTO
```

Results

XML_F52E2B61-18A1-11d1-B105-00805F49916B

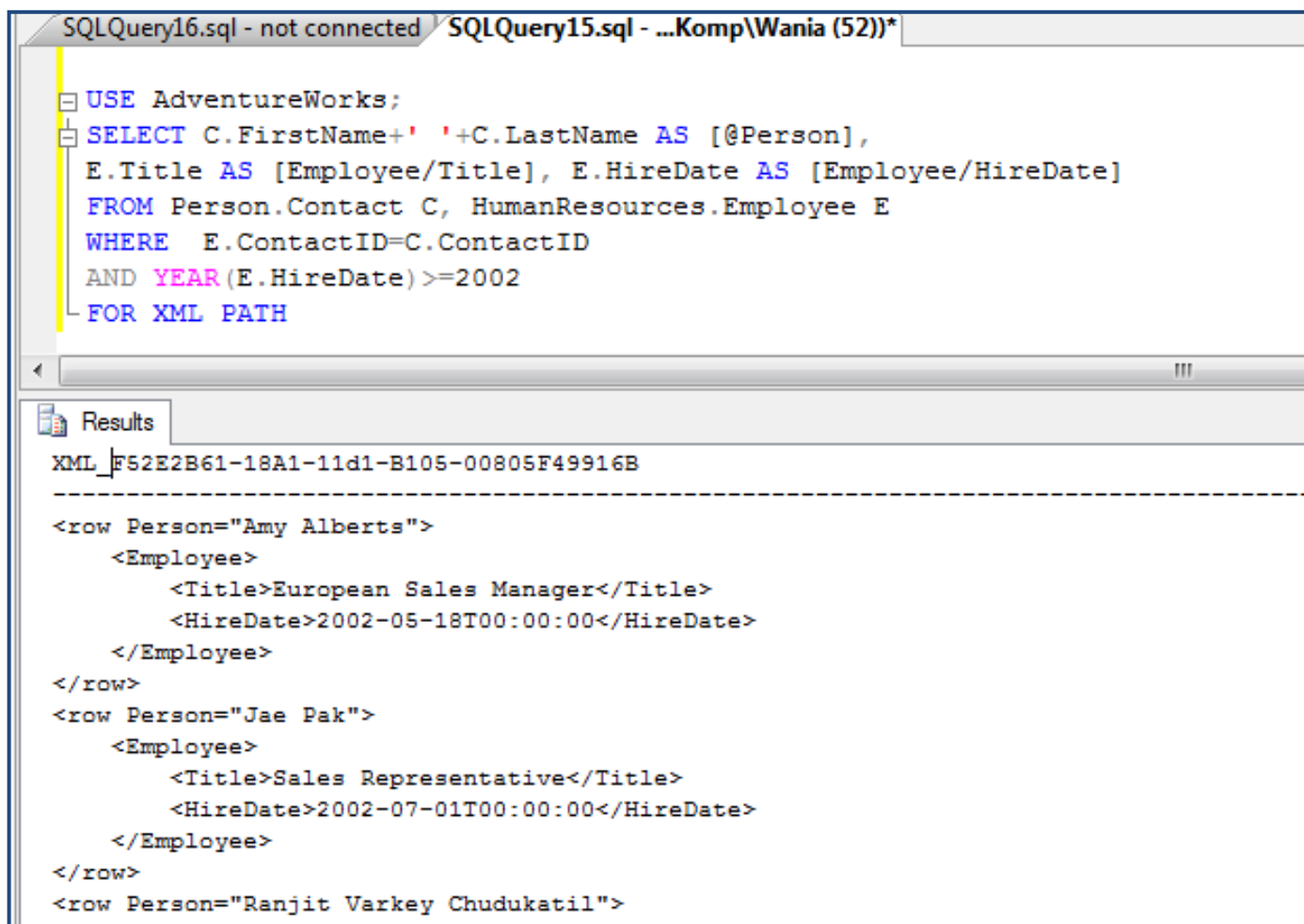
```
<Person FirstName="Rachel" LastName="Valdez">
  <Employee Title="Sales Representative" HireDate="2003-07-01T00:00:00"/>
</Person>
<Person FirstName="Lynn" LastName="Tsoflias">
  <Employee Title="Sales Representative" HireDate="2003-07-01T00:00:00"/>
</Person>
<Person FirstName="Syed" LastName="Abbas">
  <Employee Title="Pacific Sales Manager" HireDate="2003-04-15T00:00:00"/>
</Person>
<Person FirstName="Tete" LastName="Mensa-Annan">
  <Employee Title="Sales Representative" HireDate="2002-11-01T00:00:00"/>
</Person>
<Person FirstName="Jae" LastName="Pak">
  <Employee Title="Sales Representative" HireDate="2002-07-01T00:00:00"/>
</Person>
```

Dane XML

Generowanie dokumentów XML – FOR XML PATH

Strona 94

Przykład wygenerowania dokumentu XML z wybranymi danymi pracowników testowej bazy *AdventureWorks*:



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery16.sql - not connected' and 'SQLQuery15.sql - ...Komp\Wania (52))*'. The active tab displays a SQL query that uses the AdventureWorks database and selects employee data for hire dates on or after 2002, formatted for XML output using FOR XML PATH. Below the query, the 'Results' pane shows the generated XML document, which is an XML Schema Document (XSD) with a root element 'row' containing three employee records: Amy Alberts, Jae Pak, and Ranjit Varkey Chudukatil.

```
USE AdventureWorks;
SELECT C.FirstName+' '+C.LastName AS [@Person],
       E.Title AS [Employee/Title], E.HireDate AS [Employee/HireDate]
FROM Person.Contact C, HumanResources.Employee E
WHERE E.ContactID=C.ContactID
AND YEAR(E.HireDate)>=2002
FOR XML PATH
```

Results

XML_F52E2B61-18A1-11d1-B105-00805F49916B

```
<row Person="Amy Alberts">
  <Employee>
    <Title>European Sales Manager</Title>
    <HireDate>2002-05-18T00:00:00</HireDate>
  </Employee>
</row>
<row Person="Jae Pak">
  <Employee>
    <Title>Sales Representative</Title>
    <HireDate>2002-07-01T00:00:00</HireDate>
  </Employee>
</row>
<row Person="Ranjit Varkey Chudukatil">
```

W SQL Server operacje konwertowania danych XML do postaci relacyjnej możemy dokonać na dwa sposoby, wykorzystując:

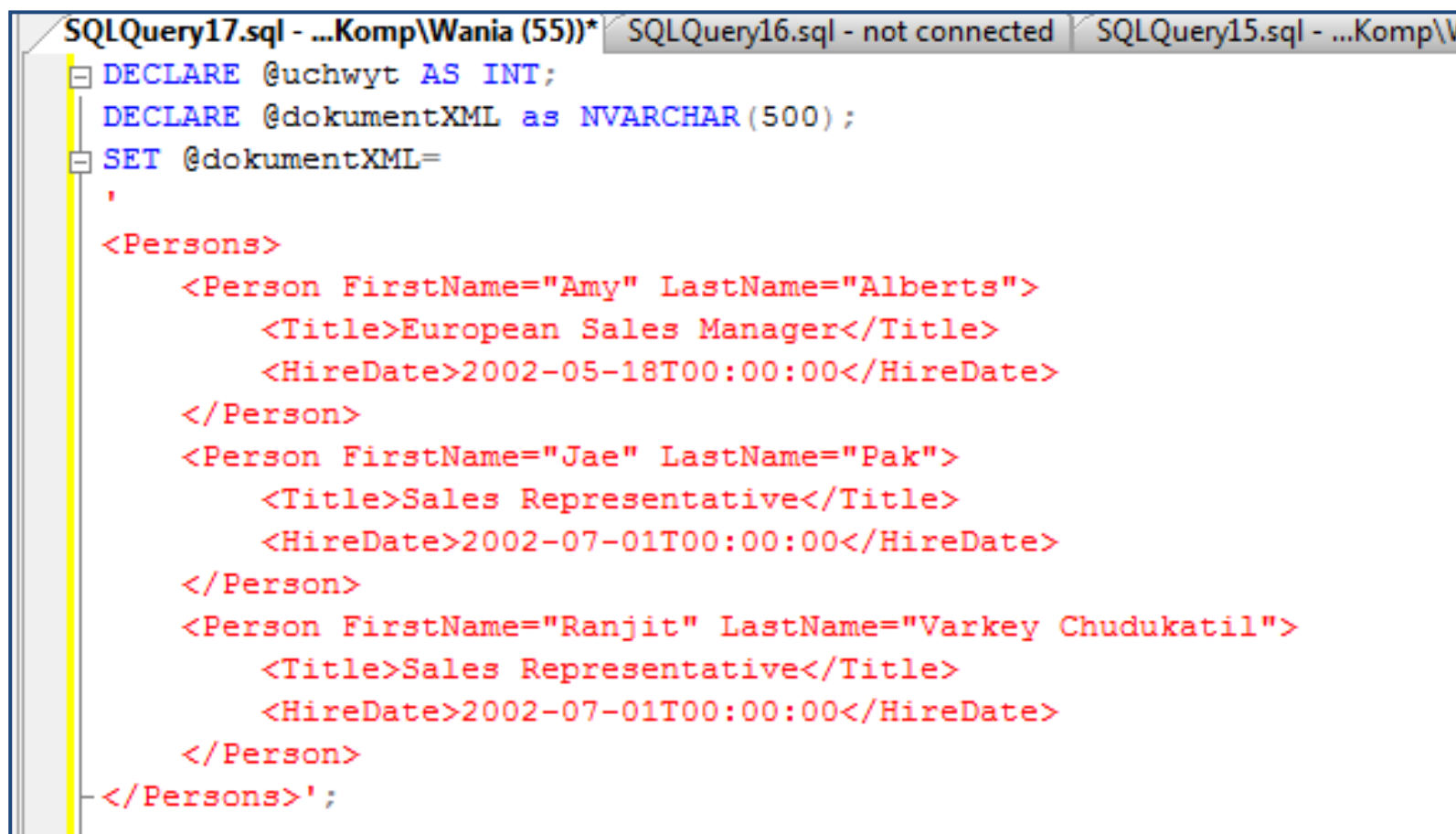
- Metodę *nodes* dla typów danych XML
- Funkcję OPENXML

Funkcja OPENXML przyjmuje następujące parametry:

- Uchwył dokumentu XML DOM (zwracany przez procedurę *sp_xml_preparedocument*)
- Wyrażenie Xpath do szukania węzłów
- Sposób mapowania między węzłami a kolumnami (flaga o możliwych wartościach 1,2 i 3)
- Opis zwracanego zbioru (klauszula WITH funkcji OPENXML)

Dane XML

Przygotowanie testowego dokumentu XML zawierającego analogiczne dane zgromadzone w atrybutach i elementach:



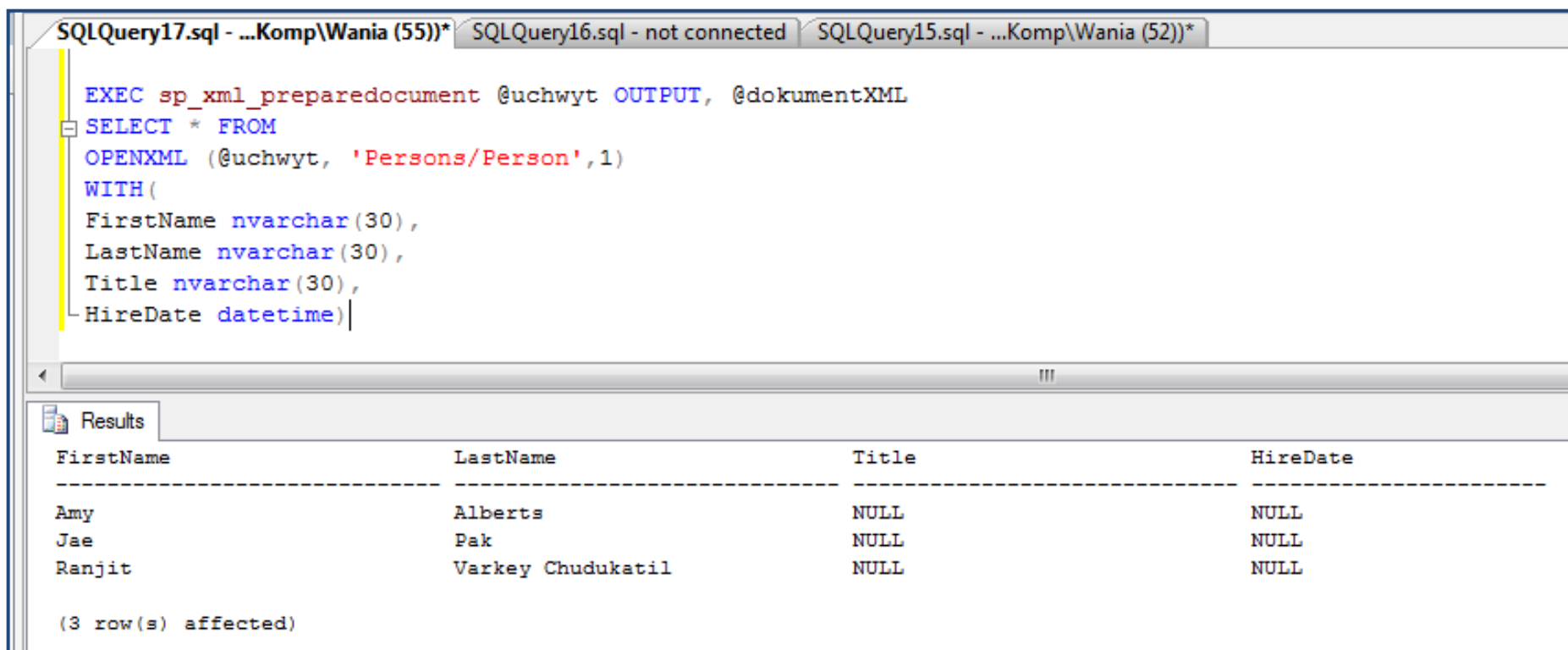
```
SQLQuery17.sql - ...Komp\Wania (55))* SQLQuery16.sql - not connected SQLQuery15.sql - ...Komp\W
DECLARE @uchwyt AS INT;
DECLARE @dokumentXML as NVARCHAR(500);
SET @dokumentXML=
'
<Persons>
  <Person FirstName="Amy" LastName="Alberts">
    <Title>European Sales Manager</Title>
    <HireDate>2002-05-18T00:00:00</HireDate>
  </Person>
  <Person FirstName="Jae" LastName="Pak">
    <Title>Sales Representative</Title>
    <HireDate>2002-07-01T00:00:00</HireDate>
  </Person>
  <Person FirstName="Ranjit" LastName="Varkey Chudukatil">
    <Title>Sales Representative</Title>
    <HireDate>2002-07-01T00:00:00</HireDate>
  </Person>
</Persons>';
```


Dane XML

Konwertowanie danych XML do postaci relacyjnej – OPENXML

Strona 97

Przykład mapowania skoncentrowanego na atrybutach testowego dokumentu XML :



The screenshot shows a SQL Server Enterprise Manager window with three tabs: 'SQLQuery17.sql - ...Komp\Wania (55))*', 'SQLQuery16.sql - not connected', and 'SQLQuery15.sql - ...Komp\Wania (52))*'. The active tab displays the following T-SQL query:

```
EXEC sp_xml_preparedocument @uchwyty OUTPUT, @dokumentXML
SELECT * FROM
OPENXML (@uchwyty, 'Persons/Person', 1)
WITH(
  FirstName nvarchar(30),
  LastName nvarchar(30),
  Title nvarchar(30),
  HireDate datetime)
```

Below the query, the 'Results' pane shows a table with four columns: 'FirstName', 'LastName', 'Title', and 'HireDate'. The table contains three rows of data:

FirstName	LastName	Title	HireDate
Amy	Alberts	NULL	NULL
Jae	Pak	NULL	NULL
Ranjit	Varkey Chudukatil	NULL	NULL

At the bottom of the results pane, it states '(3 row(s) affected)'.

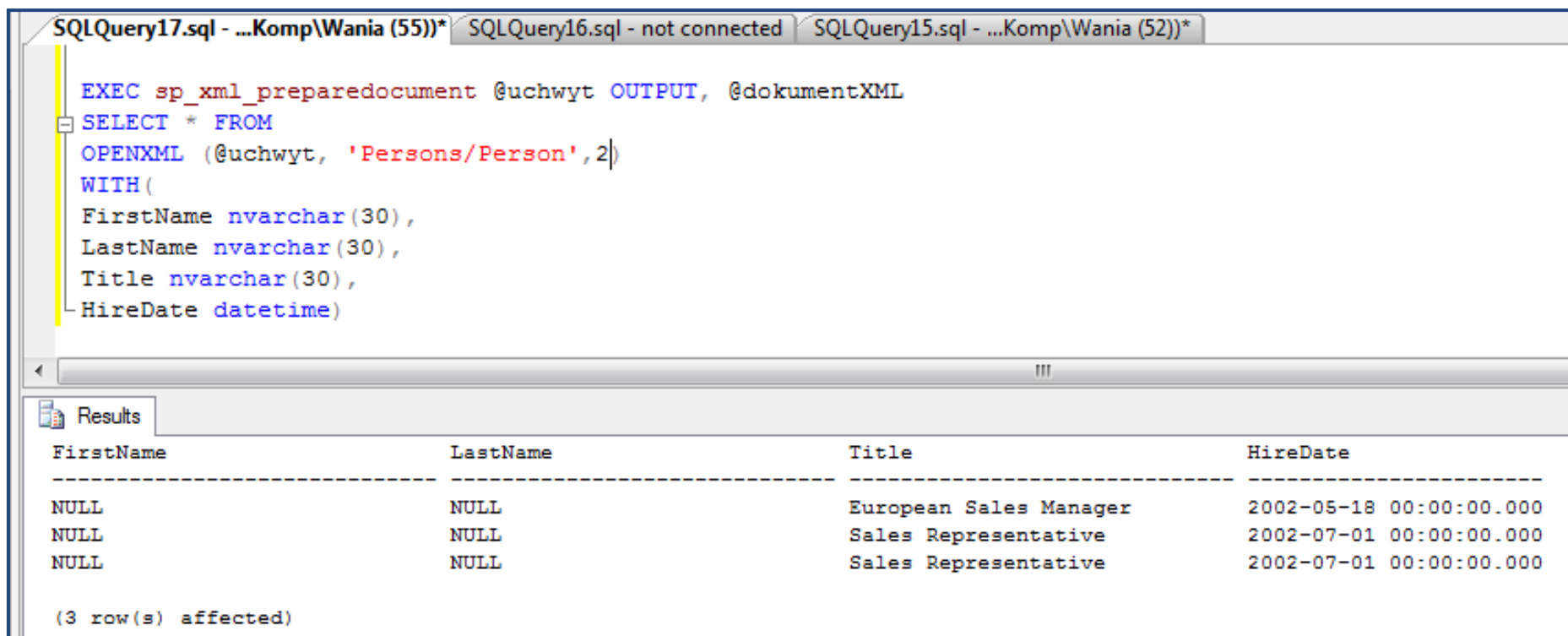
Uwaga: Kolumny tytuł i data zatrudnienia są puste ponieważ ich wartości umieszczone były w elementach dokumentu a mapowanie jest skoncentrowane tylko na atrybutach (wartość flagi 1 dla trzeciego argumentu funkcji OPENXML).

Dane XML

Konwertowanie danych XML do postaci relacyjnej – OPENXML

Strona 98

Przykład mapowania skoncentrowanego na elementach testowego dokumentu XML :



The screenshot shows a SQL Server Enterprise Manager window with three tabs: 'SQLQuery17.sql - ...Komp\Wania (55))*', 'SQLQuery16.sql - not connected', and 'SQLQuery15.sql - ...Komp\Wania (52))*'. The active tab displays the following SQL query:

```
EXEC sp_xml_preparedocument @uchwyt OUTPUT, @dokumentXML
SELECT * FROM
OPENXML (@uchwyt, 'Persons/Person', 2)
WITH (
  FirstName nvarchar(30),
  LastName nvarchar(30),
  Title nvarchar(30),
  HireDate datetime)
```

Below the query, the 'Results' pane shows a table with four columns: 'FirstName', 'LastName', 'Title', and 'HireDate'. The table contains three rows of data, all with NULL values for 'FirstName' and 'LastName'.

FirstName	LastName	Title	HireDate
NULL	NULL	European Sales Manager	2002-05-18 00:00:00.000
NULL	NULL	Sales Representative	2002-07-01 00:00:00.000
NULL	NULL	Sales Representative	2002-07-01 00:00:00.000

At the bottom of the results pane, it states '(3 row(s) affected)'.

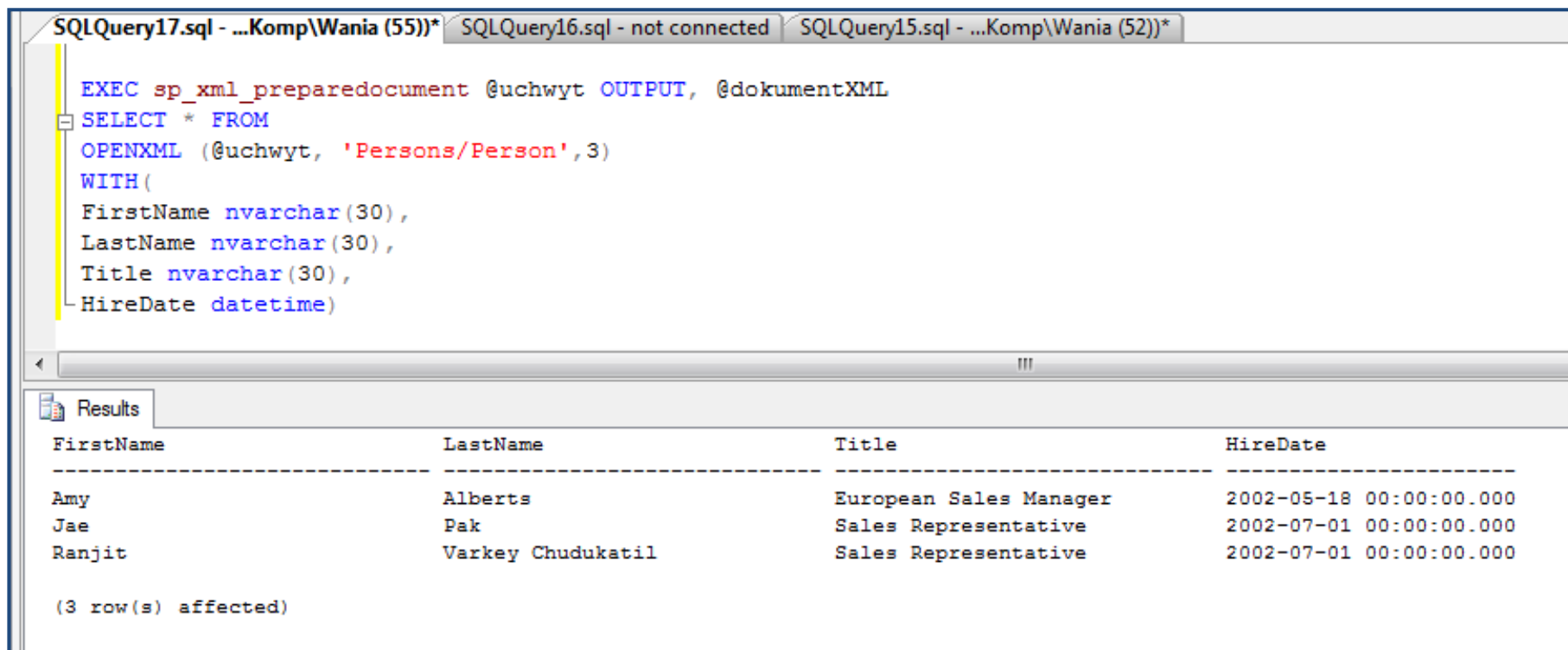
Uwaga: Kolumny imię i nazwisko są puste ponieważ ich wartości umieszczone były w atrybutach dokumentu a mapowanie jest skoncentrowane tylko na elementach (wartość flagi 2 dla trzeciego argumentu funkcji OPENXML).

Dane XML

Konwertowanie danych XML do postaci relacyjnej – OPENXML

Strona 99

Przykład mapowania skoncentrowanego na atrybutach i elementach jednocześnie dla testowego dokumentu XML :



The screenshot shows a SQL Server Enterprise Manager window with three tabs: 'SQLQuery17.sql - ...Komp\Wania (55))*', 'SQLQuery16.sql - not connected', and 'SQLQuery15.sql - ...Komp\Wania (52))*'. The active tab contains the following SQL query:

```
EXEC sp_xml_preparedocument @uchwyt OUTPUT, @dokumentXML
SELECT * FROM
OPENXML (@uchwyt, 'Persons/Person', 3)
WITH(
  FirstName nvarchar(30),
  LastName nvarchar(30),
  Title nvarchar(30),
  HireDate datetime)
```

Below the query, the 'Results' tab is active, displaying a table with four columns: 'FirstName', 'LastName', 'Title', and 'HireDate'. The table contains three rows of data:

FirstName	LastName	Title	HireDate
Amy	Alberts	European Sales Manager	2002-05-18 00:00:00.000
Jae	Pak	Sales Representative	2002-07-01 00:00:00.000
Ranjit	Varkey Chudukatil	Sales Representative	2002-07-01 00:00:00.000

At the bottom of the results, it states '(3 row(s) affected)'.

Uwaga: Wszystkie kolumny są wypełnione ponieważ mapowanie jest skoncentrowane na atrybutach i elementach jednocześnie (wartość flagi 3 dla trzeciego argumentu funkcji OPENXML).

Podstawowe właściwości:

- XQuery – standardowy język do przeglądania i pobierania danych typu XML
- XQuery – zwraca sekwencje danych które mogą być proste lub złożone zawierające węzły XML
- Nie wszystkie funkcje XQuery są wspierane przez SQL Server
- T-SQL wspiera także niestandardowe rozszerzenia języka XQuery – modyfikowanie elementów i atrybutów w danych typu XML
- query() – podstawowa metoda pobierania danych

Uwaga: Zarówno XML jak i XQuery uwzględnia wielkość liter!

Dane XML

Wykorzystanie XQuery - przykład

Strona 101

Pobieranie danych z wykorzystaniem metody query():

```
DECLARE @dokumentXML as XML;
SET @dokumentXML=
'<Persons>
  <Person FirstName="Amy" LastName="Alberts">
    <Title>European Sales Manager</Title>
    <HireDate>2002-05-18T00:00:00</HireDate>
  </Person>
</Persons>';

SELECT @dokumentXML.query('*') AS dane;
SELECT @dokumentXML.query('data(*)') AS dane;
SELECT @dokumentXML.query('data(Persons/Person/Title)') AS dane;
```

Results

dane	

<Persons><Person FirstName="Amy" LastName="Alberts"><Title>European Sales Manager</Title><HireDate>2002-05-18T00:00:00</HireDate></Person></Persons>	
(1 row(s) affected)	
dane	

European Sales Manager2002-05-18T00:00:00	
(1 row(s) affected)	
dane	

European Sales Manager	
(1 row(s) affected)	

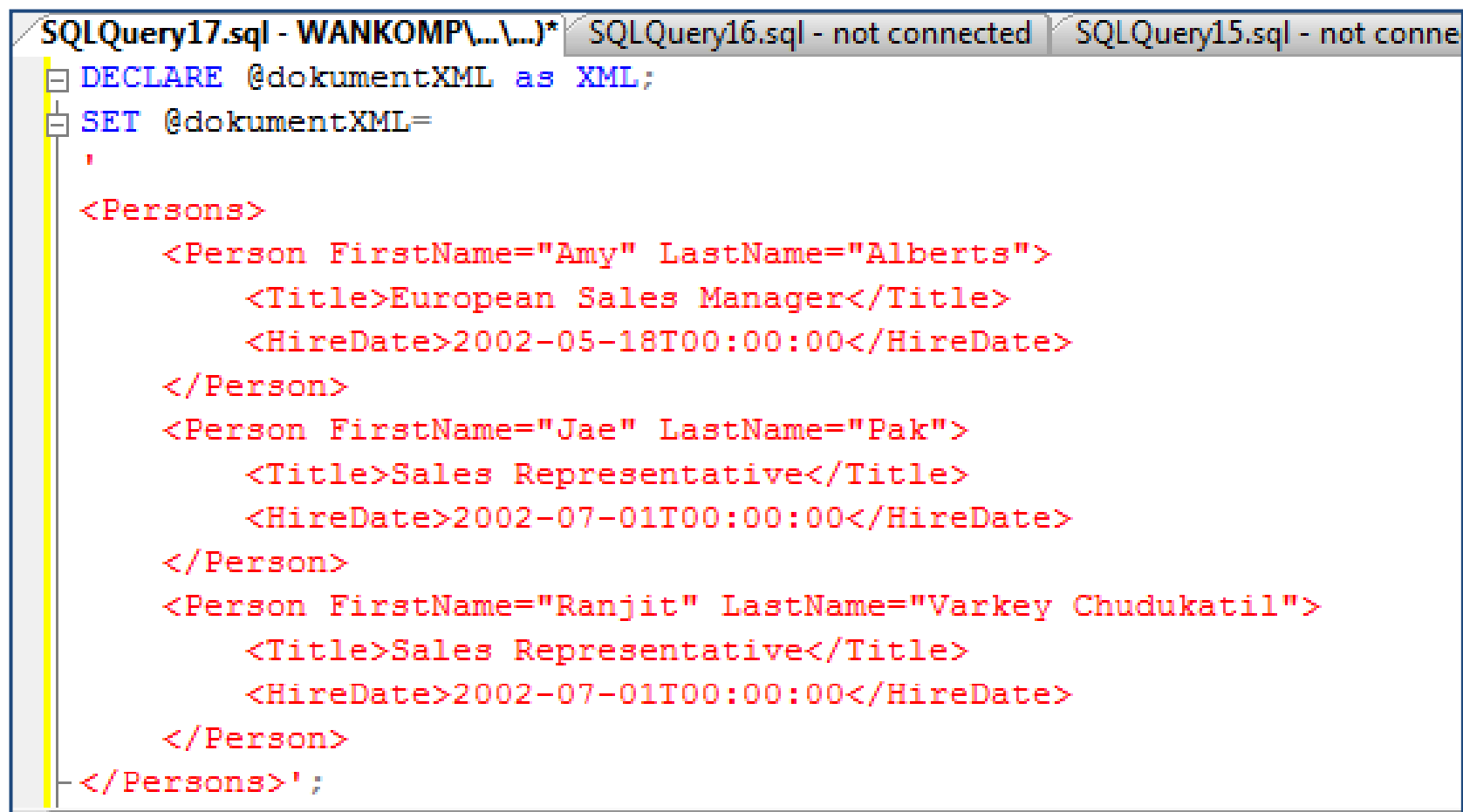
Podstawową metodę nawigacji po dokumencie XML z wykorzystaniem XQuery stanowią wyrażenia XPath .

Każda ścieka Xquery składa się z sekwencji kroków, natomiast każdy krok może składać się z trzech części:

- Określenie kierunku nawigacji – w hierarchii w górę i w dół, w prawo dla aktualnego poziomu, w aktualnym wierszu
- Testu węzłów – pobieranie węzłów o danej nazwie
- Predykatu zawężającego operację wyszukiwania – wybieranie atrybutów o określonej wartości

Dane XML

Przygotowanie testowej zmiennej typu XML:



```
SQLQuery17.sql - WANKOMP\...\...)* SQLQuery16.sql - not connected SQLQuery15.sql - not conne
DECLARE @dokumentXML as XML;
SET @dokumentXML=
'
<Persons>
  <Person FirstName="Amy" LastName="Alberts">
    <Title>European Sales Manager</Title>
    <HireDate>2002-05-18T00:00:00</HireDate>
  </Person>
  <Person FirstName="Jae" LastName="Pak">
    <Title>Sales Representative</Title>
    <HireDate>2002-07-01T00:00:00</HireDate>
  </Person>
  <Person FirstName="Ranjit" LastName="Varkey Chudukatil">
    <Title>Sales Representative</Title>
    <HireDate>2002-07-01T00:00:00</HireDate>
  </Person>
</Persons>';
```

Dane XML

Przykłady nawigacji w ścieżkach XQuery:

```
SELECT @dokumentXML.query('Persons/Person/node()') AS wezly_dzieci;
SELECT @dokumentXML.query('Persons/Person[@LastName="Pak"]/Title') AS tytul_dla_atrybutu;
```

Results

wezly_dzieci

<Title>European Sales Manager</Title><HireDate>2002-05-18T00:00:00</HireDate><Title>Sales Representative</Title>

(1 row(s) affected)

tytul_dla_atrybutu

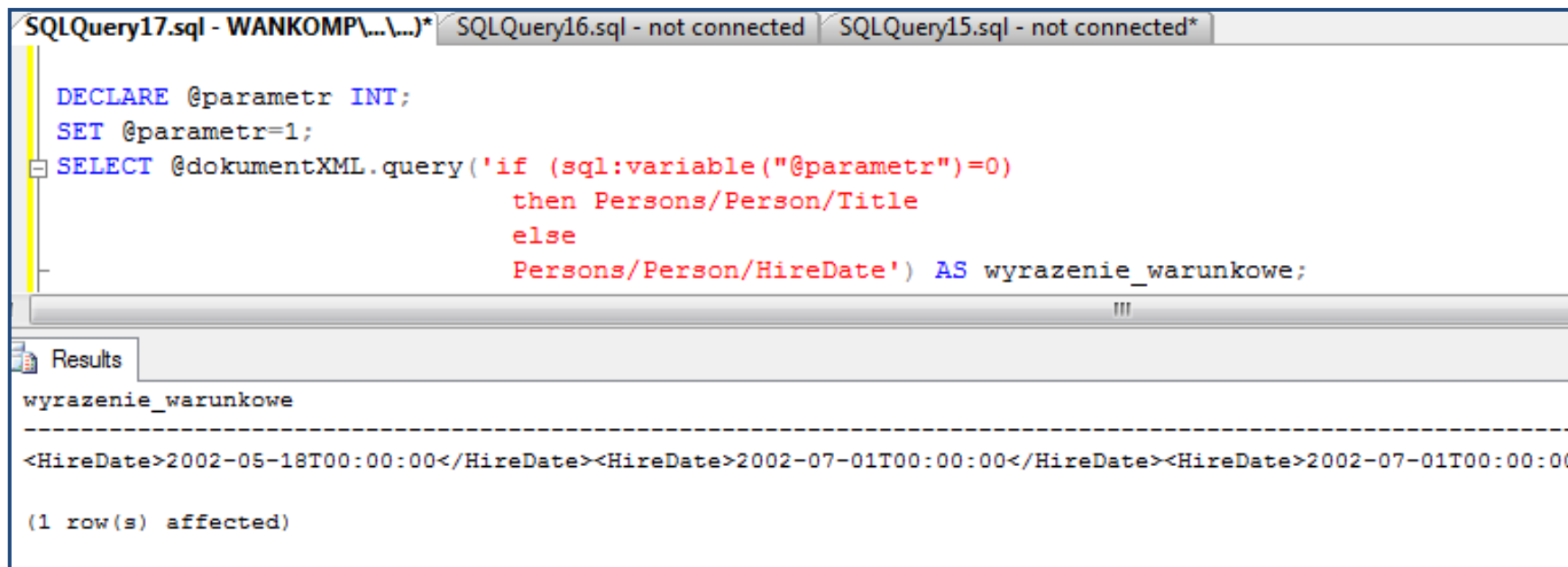
<Title>Sales Representative</Title>

(1 row(s) affected)

Zwracane są węzły, a sprawdzamy wartość atrybutów.

Dane XML

Wykorzystanie instrukcji *if...then...else*:



The screenshot shows a SQL Server Enterprise Manager window with three tabs: 'SQLQuery17.sql - WANKOMP\...\...', 'SQLQuery16.sql - not connected', and 'SQLQuery15.sql - not connected*'. The active tab contains the following T-SQL code:

```
DECLARE @parametr INT;  
SET @parametr=1;  
SELECT @dokumentXML.query('if (sql:variable("@parametr")=0)  
    then Persons/Person/Title  
    else  
    Persons/Person/HireDate') AS wyrazenie_warunkowe;
```

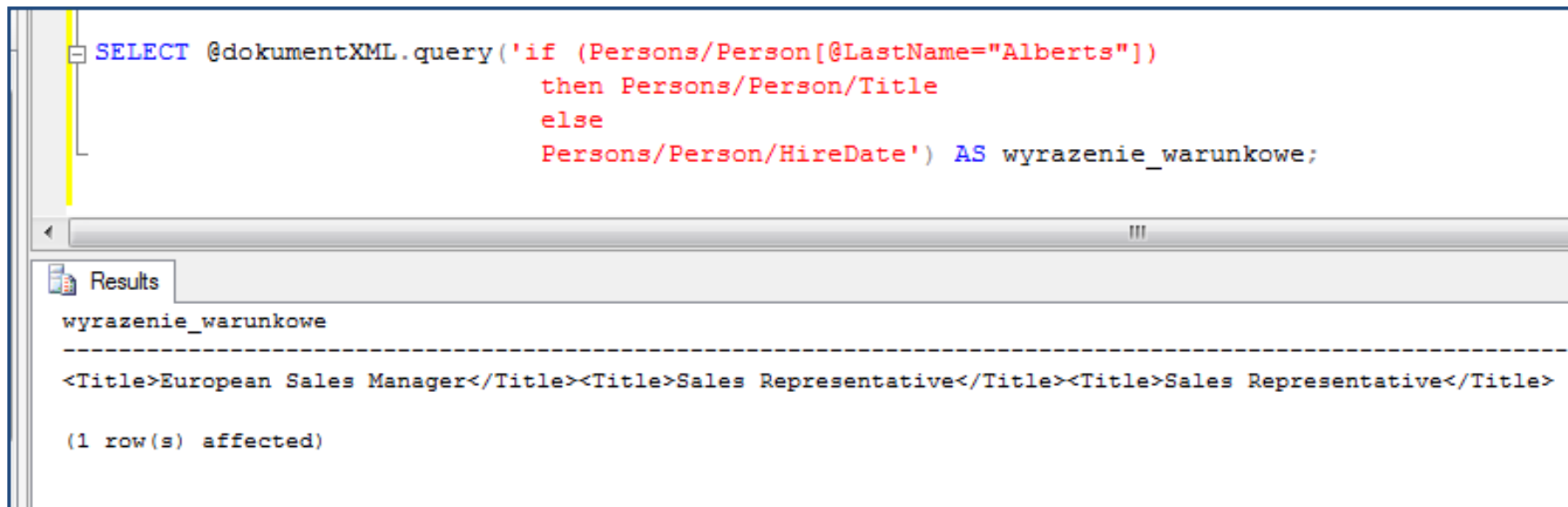
Below the query editor, the 'Results' tab is selected, displaying the XML output of the query:

```
wyrazenie_warunkowe  
-----  
<HireDate>2002-05-18T00:00:00</HireDate><HireDate>2002-07-01T00:00:00</HireDate><HireDate>2002-07-01T00:00:00  
(1 row(s) affected)
```

Z wykorzystaniem dodatkowych zmiennych T-SQL możemy decydować o przepływie zapytania XQuery

Dane XML

Wykorzystanie instrukcji *if...then...else*:



```
SELECT @dokumentXML.query('if (Persons/Person[@LastName="Alberts"])
    then Persons/Person/Title
    else
    Persons/Person/HireDate') AS wyrażenie_warunkowe;
```

Results

wyrażenie_warunkowe

<Title>European Sales Manager</Title><Title>Sales Representative</Title><Title>Sales Representative</Title>

(1 row(s) affected)

Uwaga: Operacja *if...then...else* **NIE** zmienia przepływu XQuery na podstawie wartości atrybutów! Przypomina nieco instrukcję CASE języka T-SQL.

Możliwości XQuery doskonale obrazują wyrażenia FLWOR (ang. *for, let, where, order by, return*).

Stanowi odpowiednik pętli *for each* w standardowych językach programowania.

Wyrażenia te stosowane są zazwyczaj na sekwencji węzłów.

FOR – powiązywanie zmiennych iteracji z sekwencjami wejściowymi

LET – przypisywanie wartości zmiennej dla danej iteracji

WHERE – filtrowanie danych

ORDER BY – kontrolowanie kolejności

RETURN – z każdą iteracją zostają zwrócone wyniki do systemu

Przygotowanie testowego dokumentu XML:

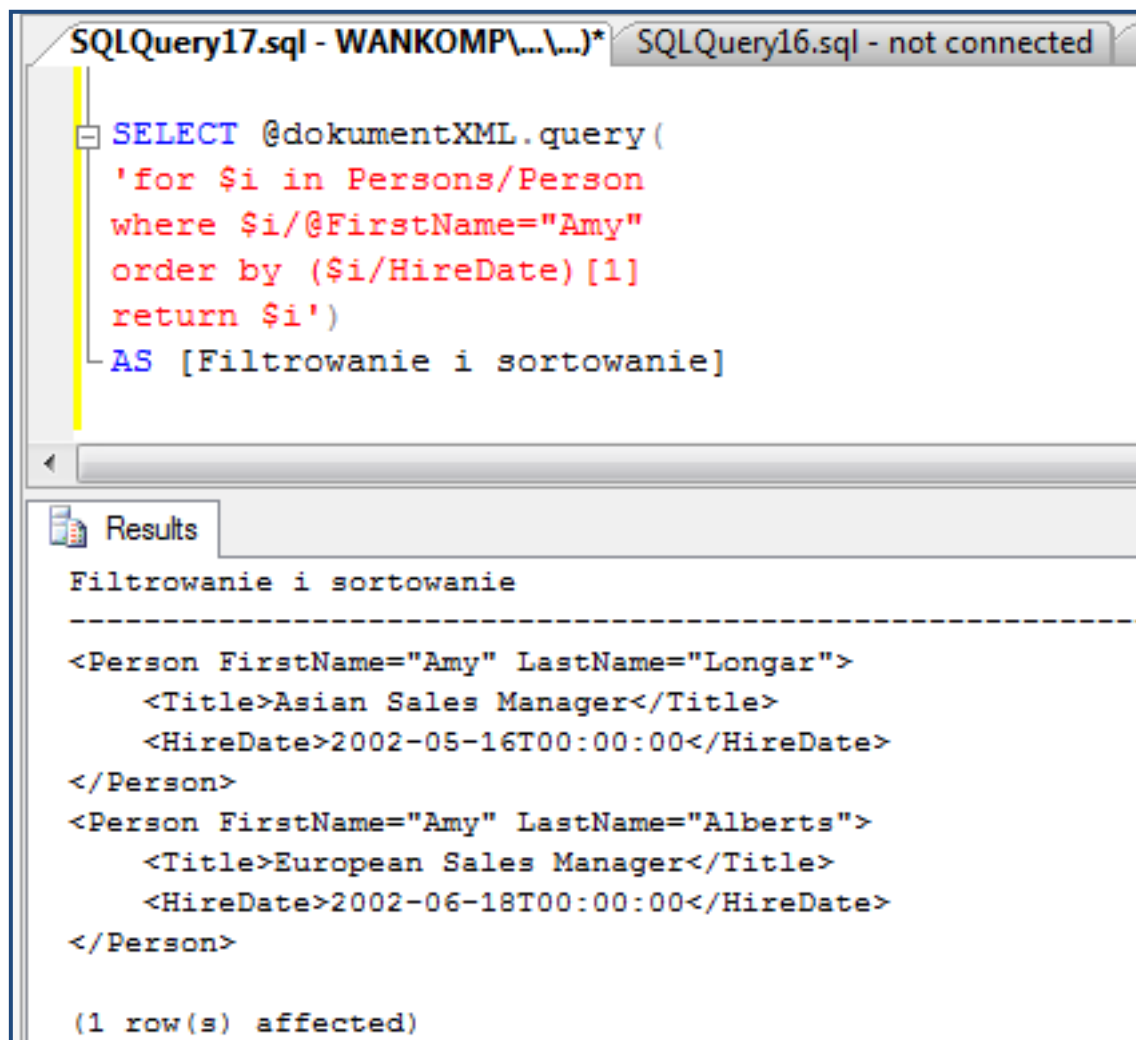
```
SQLQuery17.sql - WANKOMPA\...\... SQLQuery16.sql - not connected SQLQuery15.sql - not connected
-- DECLARE @dokumentXML as XML;
-- SET @dokumentXML=
-- '
--     <Persons>
--         <Person FirstName="Amy" LastName="Alberts">
--             <Title>European Sales Manager</Title>
--             <HireDate>2002-06-18T00:00:00</HireDate>
--         </Person>
--         <Person FirstName="Amy" LastName="Longar">
--             <Title>Asian Sales Manager</Title>
--             <HireDate>2002-05-16T00:00:00</HireDate>
--         </Person>
--         <Person FirstName="Jae" LastName="Pak">
--             <Title>Sales Representative</Title>
--             <HireDate>2002-07-01T00:00:00</HireDate>
--         </Person>
--         <Person FirstName="Ranjit" LastName="Varkey Chudukatil">
--             <Title>Sales Representative</Title>
--             <HireDate>2002-07-01T00:00:00</HireDate>
--         </Person>
--     </Persons>';
```

Dane XML

Wykorzystanie XQuery – iteracja

Strona 109

Opracowany przykład filtracji i sortowania w wyrażeniu FLWOR :



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery17.sql - WANKOMP\...\...' and 'SQLQuery16.sql - not connected'. The active tab displays an XQuery FLWOR expression. Below the query, the 'Results' pane shows the XML output of the query, which filters for 'Amy' and sorts by 'HireDate'.

```
SQLQuery17.sql - WANKOMP\...\... SQLQuery16.sql - not connected
```

```
SELECT @dokumentXML.query(  
  'for $i in Persons/Person  
  where $i/@FirstName="Amy"  
  order by ($i/HireDate)[1]  
  return $i')  
AS [Filtrowanie i sortowanie]
```

Results

Filtrowanie i sortowanie

```
<Person FirstName="Amy" LastName="Longar">  
  <Title>Asian Sales Manager</Title>  
  <HireDate>2002-05-16T00:00:00</HireDate>  
</Person>  
<Person FirstName="Amy" LastName="Alberts">  
  <Title>European Sales Manager</Title>  
  <HireDate>2002-06-18T00:00:00</HireDate>  
</Person>
```

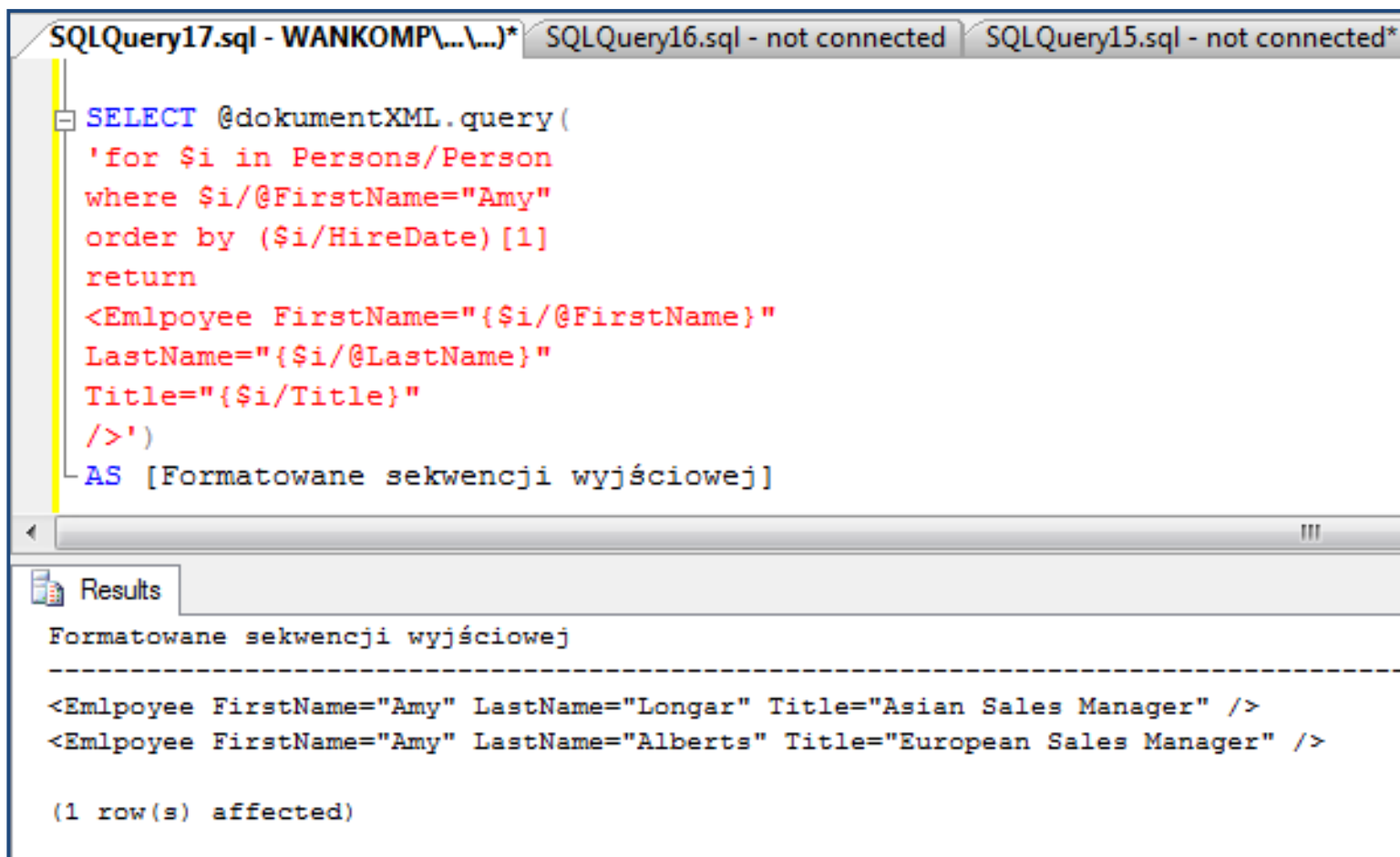
(1 row(s) affected)

Dane XML

Wykorzystanie XQuery – iteracja

Strona 110

Opracowany przykład formatowania sekwencji wyjściowej:



The screenshot shows a SQL Server Enterprise Manager window with three tabs: 'SQLQuery17.sql - WANKOMP\...\...)*', 'SQLQuery16.sql - not connected', and 'SQLQuery15.sql - not connected*'. The active tab displays an XQuery script. Below the script, the 'Results' pane shows the output of the query, which is an XML sequence of two employee records. The first record is for 'Amy Longar' with the title 'Asian Sales Manager', and the second is for 'Amy Alberts' with the title 'European Sales Manager'. The results are separated by a dashed line. At the bottom of the results pane, it indicates '(1 row(s) affected)'.

```
SQLQuery17.sql - WANKOMP\...\...)* SQLQuery16.sql - not connected SQLQuery15.sql - not connected*

SELECT @dokumentXML.query(
  'for $i in Persons/Person
  where $i/@FirstName="Amy"
  order by ($i/HireDate) [1]
  return
  <Employee FirstName="{ $i/@FirstName}"
  LastName="{ $i/@LastName}"
  Title="{ $i/Title}"
  />')
AS [Formatowane sekwencji wyjściowej]
```

Results

Formatowane sekwencji wyjściowej

<Employee FirstName="Amy" LastName="Longar" Title="Asian Sales Manager" />
<Employee FirstName="Amy" LastName="Alberts" Title="European Sales Manager" />

(1 row(s) affected)

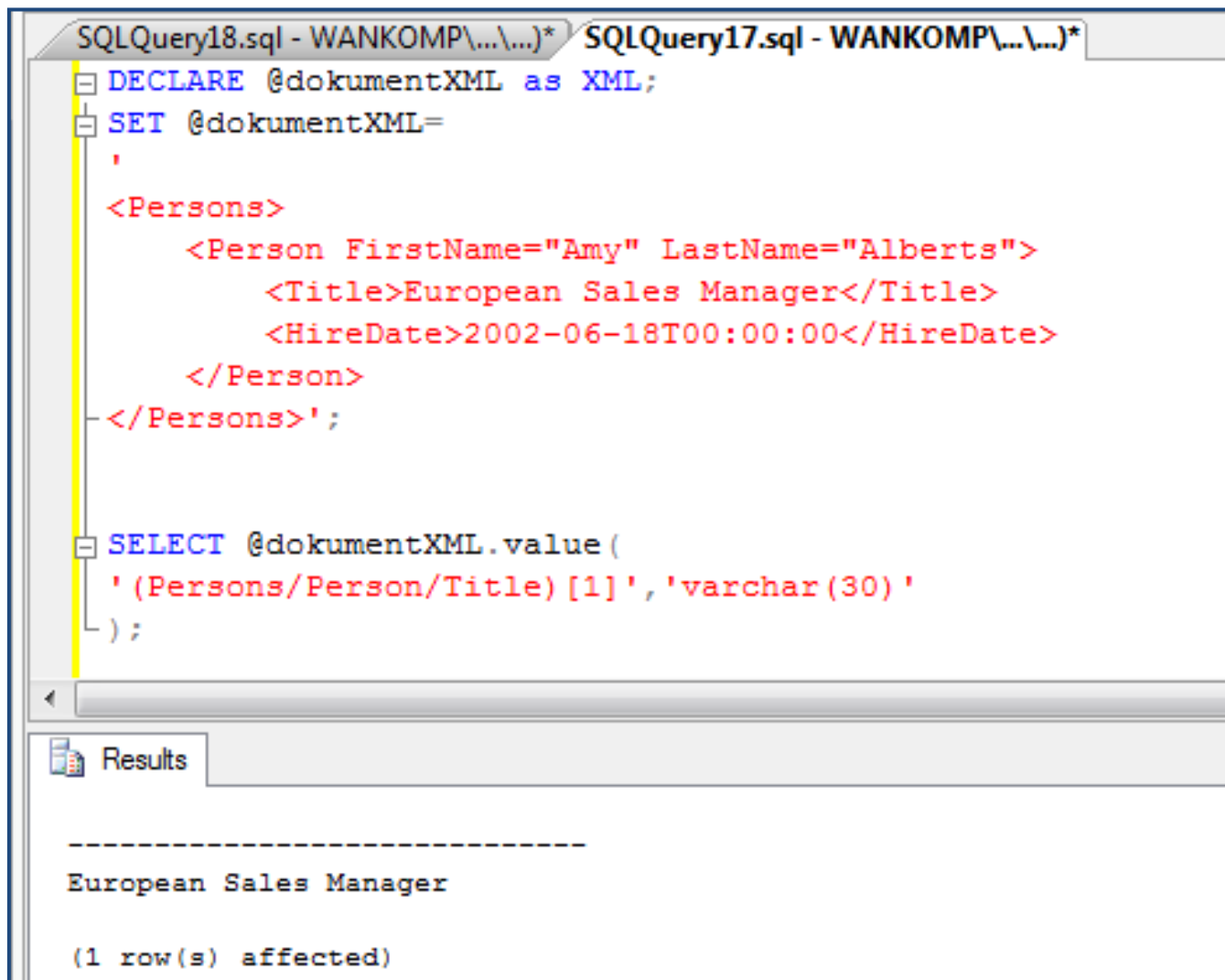
Wsparcie dla języka XML ze strony serwera bazy danych **jest istotne**, gdyż XML stanowi standardowy format danych służący do ich wymiany pomiędzy różnymi platformami i aplikacjami.

Dedykowany typ danych XML w SQL Server oferuje 5 metod, które akceptują wyrażenia Xquery:

- *query* – demonstrowana wcześniej
- *value* – pobieranie pojedynczych wartości
- *exists* – sprawdzanie istnienia
- *modify* – modyfikacje sekcji danych
- *nodes* – szatkowanie do wielu wierszy

Dane XML

Przykład wykorzystanie funkcji *value()* dla typu danych XML:



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery18.sql - WANKOMP\...\...' and 'SQLQuery17.sql - WANKOMP\...\...'. The active tab is 'SQLQuery18.sql', which contains the following SQL code:

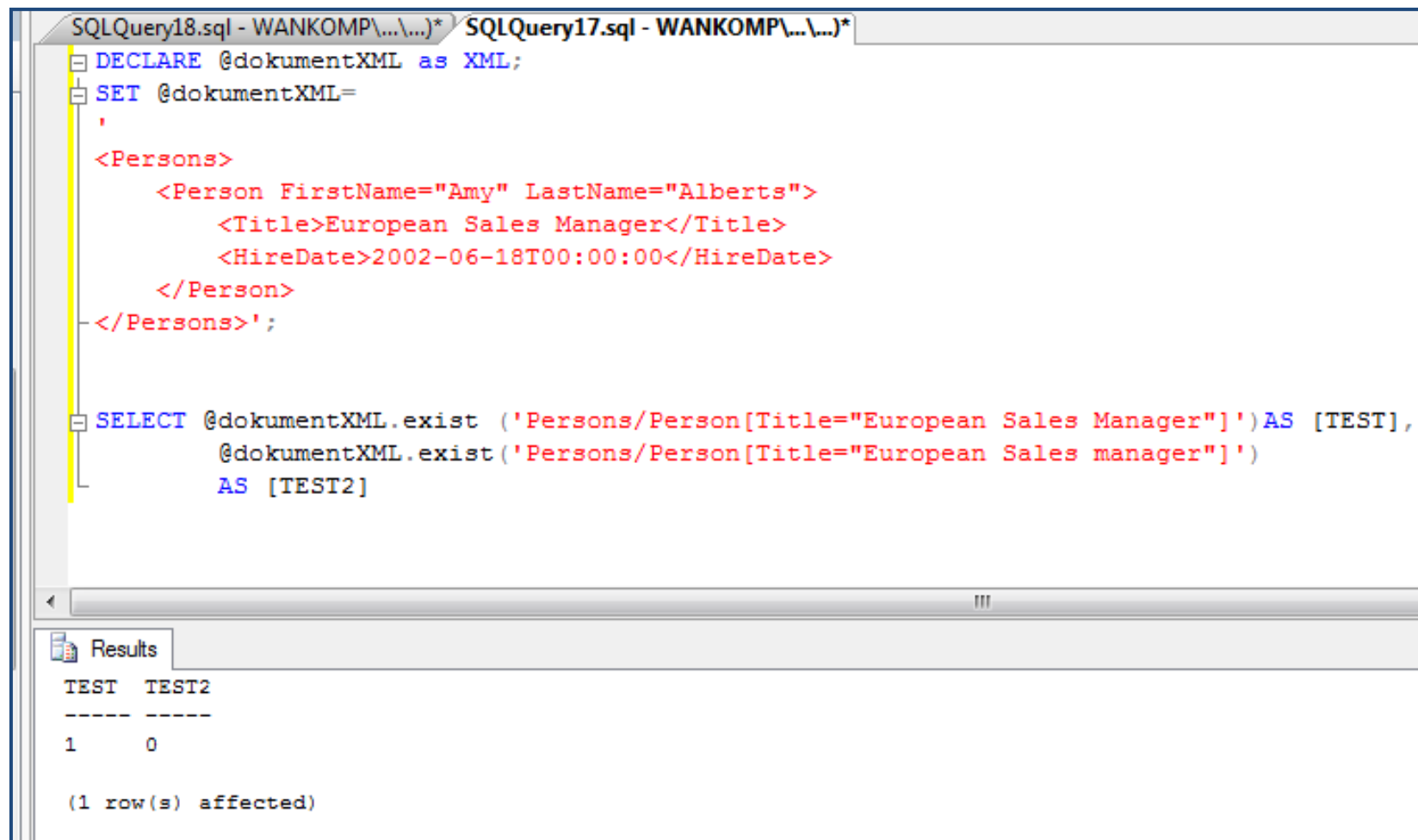
```
DECLARE @dokumentXML as XML;  
SET @dokumentXML=  
'  
<Persons>  
  <Person FirstName="Amy" LastName="Alberts">  
    <Title>European Sales Manager</Title>  
    <HireDate>2002-06-18T00:00:00</HireDate>  
  </Person>  
</Persons>';  
  
SELECT @dokumentXML.value(  
  '(Persons/Person/Title) [1]', 'varchar(30)'  
) ;
```

Below the query editor, the 'Results' pane shows the output of the query:

```
-----  
European Sales Manager  
  
(1 row(s) affected)
```


Dane XML

Przykład wykorzystanie funkcji *exists()* dla typu danych XML:



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery18.sql - WANKOMP\...\...' and 'SQLQuery17.sql - WANKOMP\...\...'. The active tab is 'SQLQuery18.sql', which contains the following T-SQL code:

```
DECLARE @dokumentXML as XML;  
SET @dokumentXML=  
'  
  <Persons>  
    <Person FirstName="Amy" LastName="Alberts">  
      <Title>European Sales Manager</Title>  
      <HireDate>2002-06-18T00:00:00</HireDate>  
    </Person>  
  </Persons>';  
  
SELECT @dokumentXML.exist ('Persons/Person[Title="European Sales Manager"]') AS [TEST],  
       @dokumentXML.exist ('Persons/Person[Title="European Sales manager"]')  
       AS [TEST2]
```

Below the query editor, the 'Results' pane shows the output of the query:

TEST	TEST2
1	0

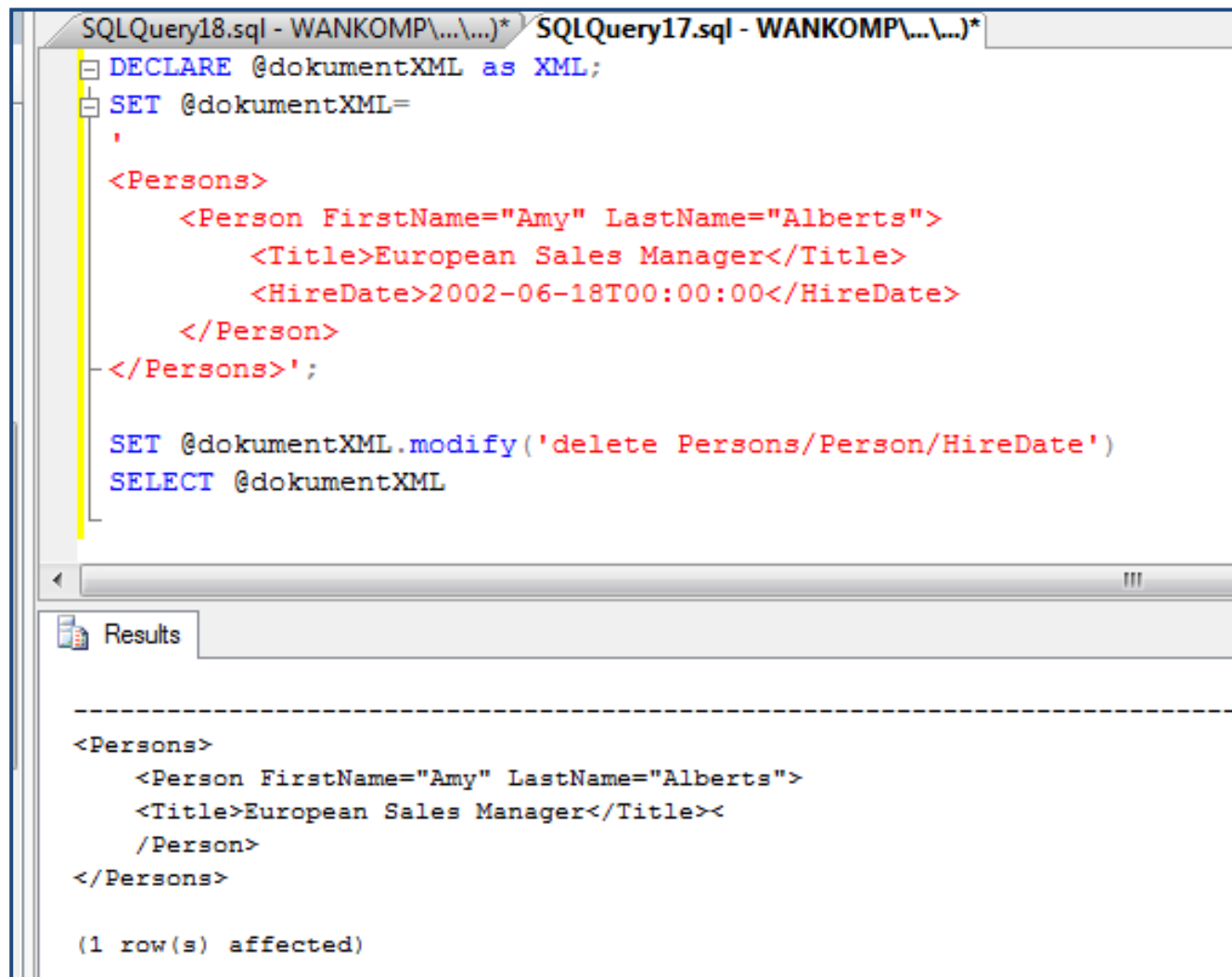
(1 row(s) affected)

Dane XML

Typ danych XML w systemie zarządzania bazą danych

Strona 114

Przykład wykorzystanie funkcji *modify()* dla typu danych XML:



The screenshot displays a SQL Server Enterprise Manager window with two tabs: 'SQLQuery18.sql - WANKOMP\...\...' and 'SQLQuery17.sql - WANKOMP\...\...'. The active tab shows a T-SQL query that declares an XML variable, sets it to a sample XML document, and then uses the `modify()` function to delete the `HireDate` element from the XML. The results pane below shows the modified XML, where the `HireDate` element has been removed. The status bar indicates '(1 row(s) affected)'.

```
SQLQuery18.sql - WANKOMP\...\... SQLQuery17.sql - WANKOMP\...\...
-- DECLARE @dokumentXML as XML;
-- SET @dokumentXML=
-- '
--     <Persons>
--         <Person FirstName="Amy" LastName="Alberts">
--             <Title>European Sales Manager</Title>
--             <HireDate>2002-06-18T00:00:00</HireDate>
--         </Person>
--     </Persons>';
--
-- SET @dokumentXML.modify('delete Persons/Person/HireDate')
-- SELECT @dokumentXML
```

Results

```
-----
<Persons>
  <Person FirstName="Amy" LastName="Alberts">
    <Title>European Sales Manager</Title><
  /Person>
</Persons>

(1 row(s) affected)
```

- Nie należy wykorzystywać wsparcia XML do przekształcania relacyjnych baz danych w „bazy XML”
- SQL Server oferuje możliwość tworzenia dokumentó XML na podstawie danych relacyjnych poprzez instrukcję FOR XML
- SQL Server oferuje również metody procesu odwrotnego – szatkowania danych, z wykorzystaniem funkcji OPENXML
- Zademonstrowane przykładowe wykorzystanie takich metod, jak:
 - *query*
 - *value*
 - *exists*
 - *nodes*

Wykorzystujące jako argumenty wyrażenia XQuery

- Microsoft SQL Server 2008 od środka: Programowanie w języku T-SQL, MS Press 2010
- Microsoft SQL Server 2005 od środka: Programowanie w języku T-SQL, Ms Press 2006
- Microsoft SQL Server 2005 od środka: Zapytania w języku T-SQL, MS Press 2006
- Microsoft SQL Server 2005: Rozwiązania praktyczne, MS Press 2006
- Microsoft SQL Server 2005: Podręcznik programisty, Helion 2007

Podsumowanie

Współczesne systemy zarządzania relacyjnymi bazami danych, takie jak: MS SQL Server, Oracle, Sybase SQL Anywhere, PostgreSQL, MySQL, DB2 są obecnie bardzo rozbudowane o wiele dodatkowych funkcjonalności oraz wszystkie charakteryzują się możliwością przenoszenia znacznej liczby funkcjonalności aplikacji na poziom bazy danych.

Coraz częściej jako kolejny język programowania, wymieniane są języki oparte o język SQL, takie jak: Transact SQL, pl_psql...

W związku z obecnie ogromnymi możliwościami tych języków powstała nowa specjalizacja informatyczna jak programista baz danych.



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego
Projekt PO KL 4.1.2/2009 "Inżynier pilnie poszukiwany" nr umowy: UDA-POKL.04.01.02-00-141/09-00

Zaawansowane programowanie baz danych

dr inż. Piotr Ratuszniak ratusz@ie.tu.koszalin.pl

KATEDRA INŻYNIERII KOMPUTEROWEJ

WYDZIAŁ ELEKTRONIKI I INFORMATYKI

POLITECHNIKA KOSZALIŃSKA

2009-2013