Microsoft® SQL Server™ 2008

Praktický sprievodca novinkami

Ľuboslav Lacko

2. aktualizované vydanie (SQL Server 2008 – finálna verzia)

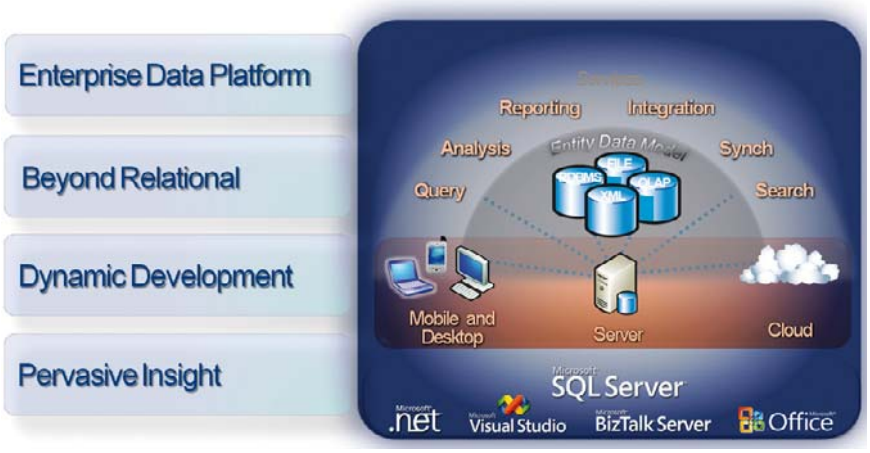
Microsoft® SQL Server™ 2008

Microsoft SQL Server 2008 praktický sprievodca novinkami

Obsah:

Kapitola 1: Úvod a inštalácia

Pod označením SQL Server 2008 sa neskrýva len samotný databázový server, ale komplexná moderná, výkonná, spoľahlivá a bezpečná serverová platforma pre ukladanie a správu údajov v databázach a údajových skladoch a balíky nástrojov pre Business Intelligence vrátane pokročilého reportovania. Uplatní sa v širokej škále podnikov všetkých veľkostí. Poradí si nielen s relačnými a multidimenzionálnymi údajmi, ale poskytne spoľahlivé úložisko aj pre multimediálne dokumenty a súbory. Nová črta „spatial" poskytuje široké možnosti spracovania geometrických a geografických údajov, napríklad z rôznych GPS zariadení a podobne. Niektoré črty sú vylepšené z predchádzajúcej verzie, no niektoré sú úplne nové a je potrebné, aby sa s nimi vývojári databázových aplikácií zoznámili a vedeli ich efektívne využívať v nových aplikáciách.



SQL Server 2008 – komplexná a škálovateľná databázová platforma

Microsoft SQL Server 2008 ako súčasť novej vlny produktov

Databázový server SQL Server 2008 je integrálnou súčasťou novej vlny uvádzaných produktov. V tejto vlne reprezentuje vývojárske nástroje Visual Studio 2008 a s ním spojená technologická platforma .NET Framework 3.5. Komerčná verzia je na trhu od decembra 2007. V roku 2008 pribudol do rodiny serverov z dielne spoločnosti Microsoft nový operačný systém Windows Server 2008 a SQL Server 2008. Aby bolo možné vyvíjať aplikácie pre SQL Server 2008, je potrebné doplniť Visual Studio 2008 o balíček SP1.

SQL Server 2008 je jedným z nosných pilierov novej „vlny" produktov spoločnosti Microsoft

Pre nainštalovanie Service Packu SP1 pre Visual Studio 2008 môžeme stiahnuť krátky pol megabajtový zavádzací súbor, ktorý následne načíta inštalačné súbory z internetu

http://www.microsoft.com/downloads/
details.aspx?familyid=FBEE1648–7106–44A7–9649–6D9F6D58056E&displaylang=en

alebo 800 MB kompletný ISO obraz inštalačného média na adrese

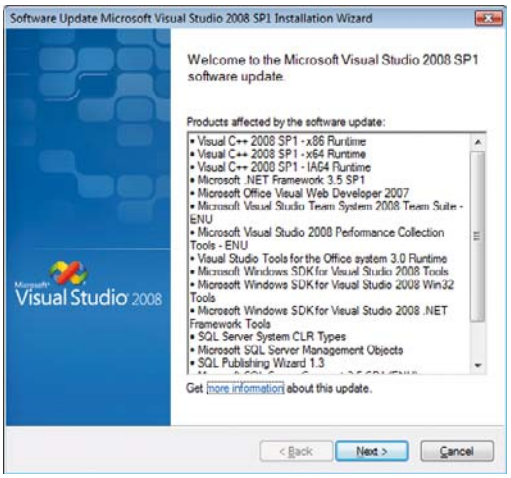
http://www.microsoft.com/downloads/
details.aspx?familyid=27673C47-B3B5–4C67-BD99–84E525B5CE61&displaylang=en

Visual Studio 2008 SP1 ponúka zlepšené nástroje na tvorbu WPF aplikácií, plnú podporu SQL Servera 2008, ADO.NET Entity Framework Designer a nástroje na použitie ADO.NET Data Services, doplnkové nástroje a komponenty Visual Basic a Visual C++. Balíček prináša ja vylepšenia ohľadne vývoja a nasadenia webov s bohatšou podporou JavaScriptu, posilnenými AJAX a dátovými nástrojmi.

Service Pack1 pre Visual Studio 2008 obsahuje aj doplnok NET Framework 3.5 SP1, ktorý v niektorých aplikáciách založených na WPF (Windows Presentation Foundation) môže poskytnúť vyšší výkon o 20 až 45 % bez nutnosti zmeny kódu. Vylepšenia vo WCF poskytnú vývojárom viac kontroly nad spôsobom, akým pristupujú k dátam a službám. NET Framework 3.5 SP1 umožňuje aj distribúciu.NET Frameworku pre klientské aplikácie s optimalizovanou veľkosťou.

Z ďalších čŕt a funkcií, ktoré sú zapuzdrené v SP1 pre Visual Studio 2008 námatkovo spomenieme

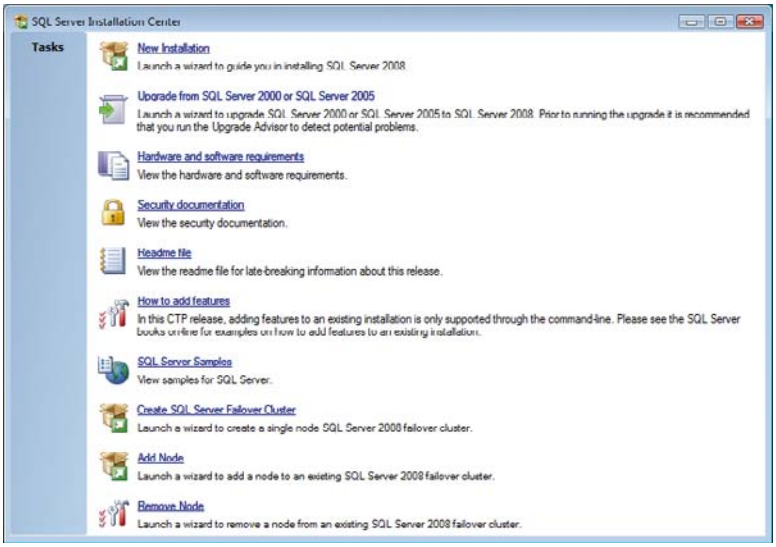
- Microsoft Office Visual Web Developer 2007
- Microsoft Visual Studio 2008 Performance Collection Tools
- Visual Studio Tools for the Office system 3.0 Runtime
- Microsoft Windows SDK for Visual Studio 2008 Tools
- SQL Server System CLR Types
- Microsoft SQL Server Management Objects
- SQL Publishing Wizard 1.3
- Microsoft SQL Server Compact 3.5 SP1



Inštalácia doplnku Service Pack1 pre Visual Studio 2008

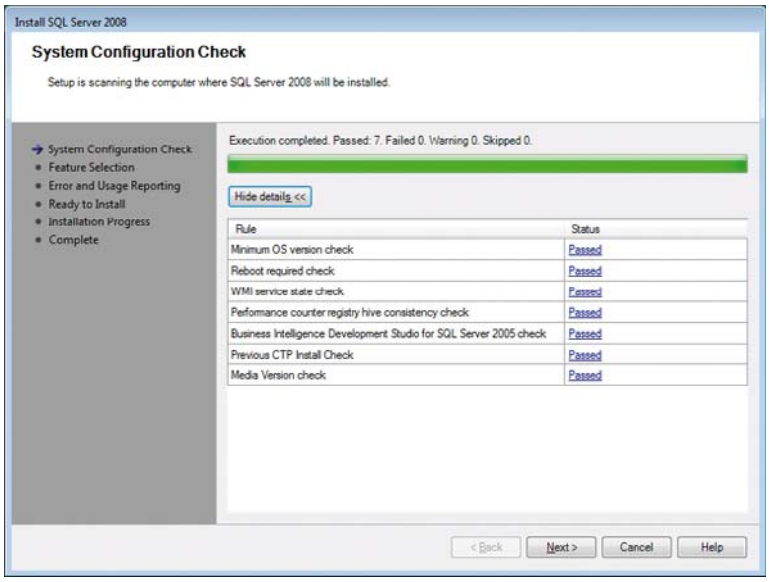
Inštalácia SQL Servera 2008

Pre praktické zoznámenie sa s novým databázovým serverom MS SQL Server 2008 bude príručka zameraná viac prakticky, to znamená, že jednotlivé témy budú preberané na cvičných príkladoch. Preto ako prvý úkon odporúčame inštaláciu. Inštalácia všetkých služieb programov a nástrojov servera SQL Server 2008 je sústredená do utility SQL Server Instalation Center. Ako veľmi zaujímavú alternatívu pre testovanie a zoznamovanie sa so serverom SQL Server 2008 odporúčame nainštalovať ho na virtuálny počítač vytvorený pomocou nástroja Microsoft Virtual PC 2007. Ako operačný systém odporúčame pre vývojárov a študentov Windows Vista a pre migrujúcich administrátorov Windows Server 2008. V prípravnej fáze sa nainštaluje technologická platforma Microsoft .NET Framework 3.5, prípadne sa upgraduje na verziu SP1 a podporné súbory (Power Shell, MSI...).



Ponuka možností SQL Server Installation Center

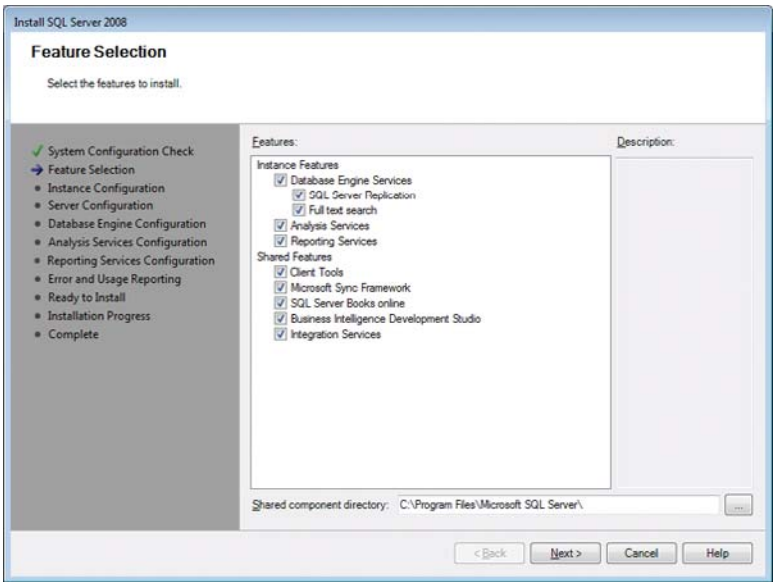
V ďalšej prvej fáze po súhlase s licenčnými podmienkami inštalačný program skontroluje konfiguráciu počítača, či je pre inštaláciu servera SQL Server 2008 vhodný. V prípade nevhodného operačného systému, malej pamäte alebo diskovej kapacity, prípadne pomalého procesora bude používateľ na túto skutočnosť upozornený.



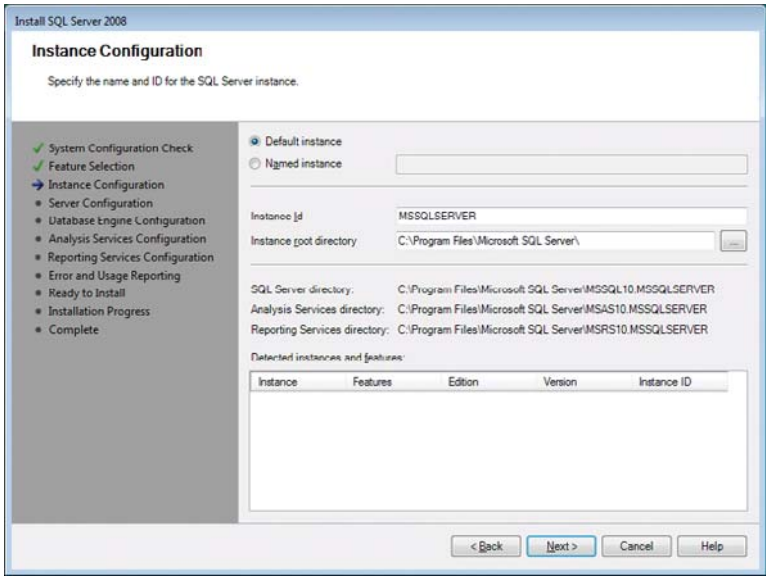
Kontrola konfigurácie systému

Po kontrole hardvérovej a softvérovej kompatibility je potrebné špecifikovať, ktoré komponenty chceme nainštalovať. V ponuke sú komponenty:

- databázové služby, v rámci nich môžeme zvoliť, či chceme nainštalovať nástroje pre replikáciu a fulltextové vyhľadávanie,
- analytické služby,
- reportovacie služby,
- integračné služby,
- administrátorské a vývojárske nástroje, Business Intelligence Development Studio, pomocné komponenty a dokumentácia...

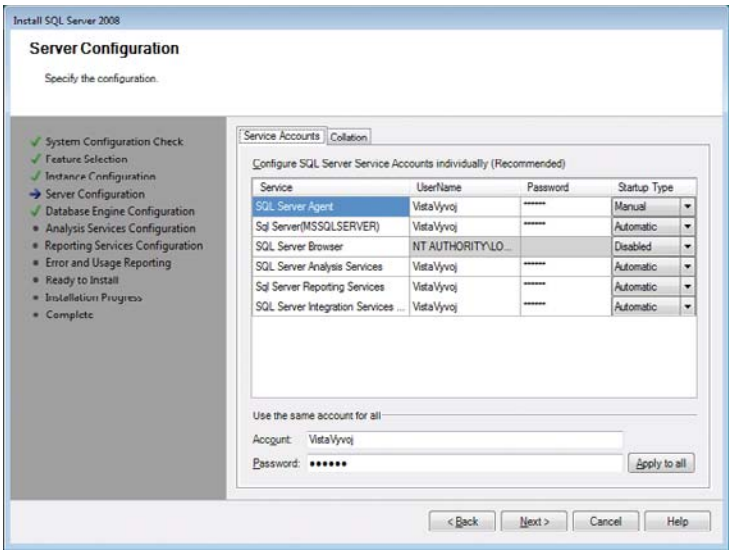


Výber komponentov pre inštaláciu



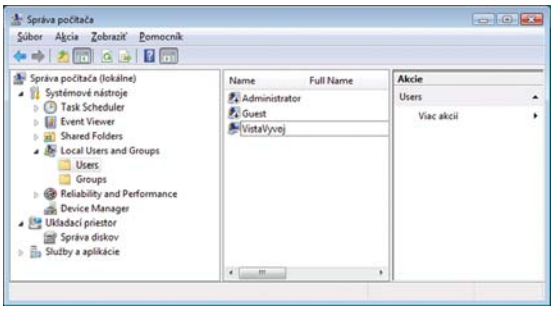
Konfigurácia inštancie

Pokračujeme nastavením prístupových účtov. Vo verzii servera SQL Server 2008 môžeme nastaviť tieto parametre pre každú požadovanú službu zvlášť. Zároveň v tomto kroku definujeme automatický štart služieb pri spustení operačného systému, prípadne zvolíme možnosť spustiť niektorú službu v prípade potreby ručne.



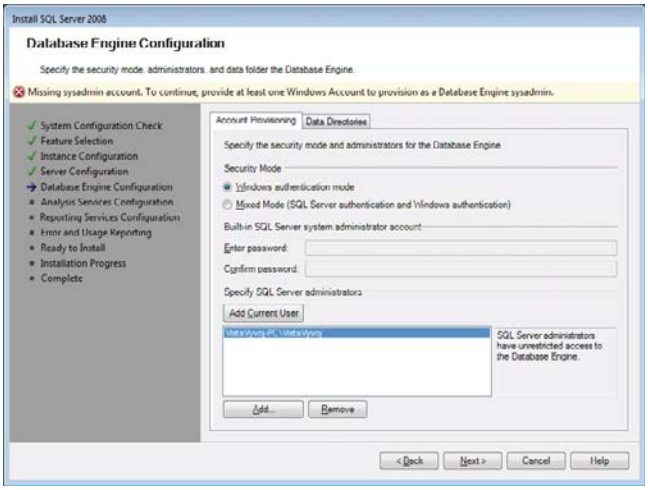
Konfigurácia servera – prístupové účty

Prístupové účty sú spoločné s operačným systémom Windows.



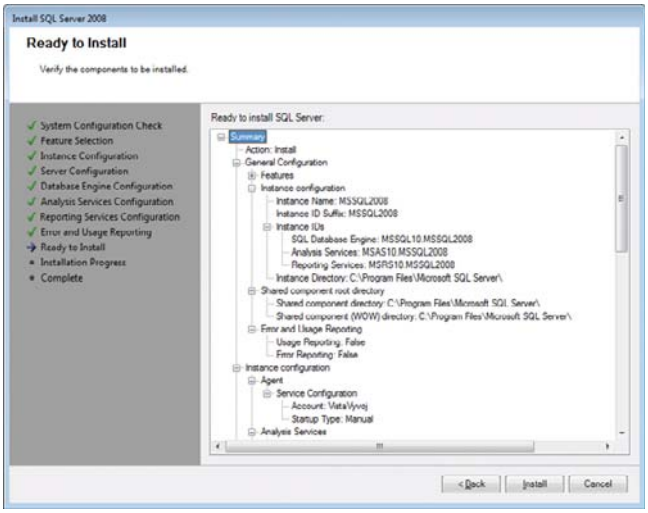
Prístupové účty sú spoločné s operačným systémom Windows

Nasleduje voľba lokálneho alebo doménového prístupu. Môžeme zvoliť možnosť *Windows Authentication Mode* alebo *Mixed Mode*, kedy sa overí autentifikácia operačného systému Windows spolu s prihlasovacím heslom servera SQL Server. Pre tento mód je potrebné zadať heslo, ktoré budeme používať ako administrátor.



Konfigurácia typu prístupu

V záverečnom resumé môžeme skontrolovať komponenty, ktoré chceme inštalovať a spustíme inštaláciu.



Záverečné resumé pred zahájením inštalácie

Kapitola 2: Konfigurácia, administrácia a vytvorenie cvičnej databázy

SQL Server 2008 je súhrnný názov pre súbor služieb na pozadí a nástrojov pre ich správu. Najdôležitejšou aplikáciou pre správu databázového servera je SQL Server Management Studio. Je to zároveň aj akýsi „priezor" pomocou ktorého môžeme zistiť, čo databázový server robí, aké štruktúry spravuje a podobne. Tento nástroj spustia pre zoznámenie sa so serverem SQL Server 2008 aj „migranti" z iných databázových platforiem a v neposlednom rade aj začiatočníci v odbore databáz, ktorí si naštudovali základy teórie databáz a jazyka SQL a budú si chcieť vyskúšať prvý jednoduchý príklad. Bude ich určite zaujímať, kam vlastne tie SQL príkazy zadávať a kde a v akej forme získajú výsledok.
SQL Server Management Studio je komplexné integrované prostredie pre správu databázového servera SQL Server 2008. Spúšťa sa štandardným spôsobom z menu operačného systému Windows *Start |All Programs | Microsoft SQL Server 2008 | SQL Server Management Studio*. Pri štandardnej inštalácii servera SQL Server 2008 v operačnom systéme Windows sa tento nástroj nachádza v priečinku:
C:\Program Files\Microsoft SQL Server\100\Tools\Binn\VSShell\Common7\IDE\Ssms.exe

Po spustení sa zobrazí dialóg pre pripojenie sa k databázovému serveru. Ak robíme zoznamovacie pokusy s novým databázovým serverom môžeme ako metódu prihlásenia použiť Windows Authentication.

Prihlasovací dialóg a voľba pripojenia nástroja Management Studio 2008 na server

Ak nepoznáme názov databázového servera, môžeme kliknúť na šípku vpravo v kolónke pre zadanie názvu servera, (Server Name) a využiť možnosť nechať si zobraziť zoznam dostupných databázových, analytických, reportovacích a integračných služieb.

Po zatlačení tlačidla Options sa sprístupní rozšírený mód prihlasovacieho dialógu so záložkami Login a Connection Properties a Additional Connection Parameters. V záložke Connection Properties sú podrobnejšie parametre pre pripojenie sa k databázovému serveru. Môžeme nastaviť typ sieťového protokolu a časový limit pre pripojenie a časový limit pre vykonanie príkazu.

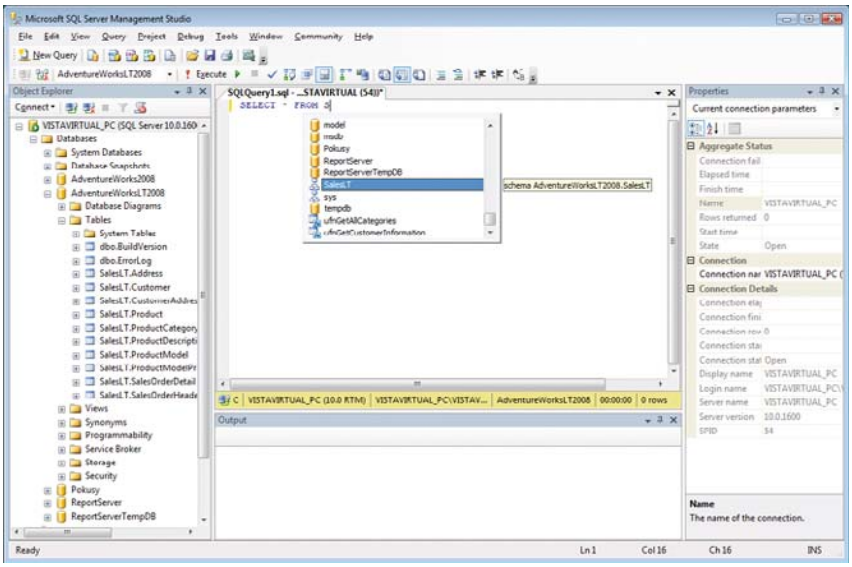
Záložka prihlasovacieho dialógu „Connection Properties"

Object Explorer
Okno poskytuje grafický, prehľadný, hierarchický pohľad na SQL komponenty až na úroveň stĺpcov a indexov. Je potrebné si uvedomiť, že objekty sú vo verzii servera SQL Server 2005 a 2008 asociované so schémou a nie s vlastníkom ako tomu bolo pri predchádzajúcej verzii SQL 2000.

Dopyty
Po aktivovaní tlačidla New Query aktivujeme dopytovací mód nástroja Management Studio. Tento mód je nástupcom nástroja Query Analyzer zo staršej verzie servera SQL Server 2000. Okrem klasických SQL a T-SQL dopytov je možné prepnúť Management Studio do viacerých režimov:

- MDX Query,
- DMX Query,
- XMLA Query,
- SQL Compact Query,

podľa toho, či pracujeme s relačnými, multidimenzionálnymi alebo data miningovými údajmi. Veľmi dobrým pomocníkom pri tvorbe dopytov nám bude črta intellisense, ktorá nám napovedá názvy funkcií objektov a podobne.



Intellisense v prostredí SQL Server Management Studio

Skôr než začneme zadávať SQL dopyty do niektorej databázy, odporúčame uistiť sa, či je Management Studio prepnuté na správnu databázu. Prepnutie sa realizuje buď pomocou combo boxu na hornom paneli nástrojov aplikácie, alebo príkazom:

```
use NazovDatabazy
```

Napríklad

```
use AdventureWorksLT2008
```

Príkazy môžeme zlučovať aj do dávok:

```
use AdventureWorksLT2008
GO

SELECT City, StateProvince, CustomerCount = count(*)
FROM SalesLT.Address
GROUP BY City, StateProvince
HAVING count(*) > 1
ORDER BY count(*) desc
GO
```

Výsledky dopytovania je možné zobraziť vo forme tabuľky alebo textového výpisu, prípadne uložiť do súboru.

V okne pre spúšťanie T-SQL dopytov môžeme testovať nielen SQL príkazy ale napríklad aj spúšťať uložené procedúry. Pre tento účel slúži príkaz EXEC. Ukážeme príklad spustenia systémovej uloženej procedúry SP_WHO, ktorá vypíše informácie o používateľoch a procesoch:

```
EXEC sp_who
GO
```

Uloženú procedúru môžeme volať aj s parametrami. Syntax parametrov pre procedúru SP_WHO:

sp_who [[@login_name =] 'login']

V konzolovej aplikácii voláme procedúru s parametrom v tvare:

```
EXEC sp_who 'VISTAVIRTUAL_PC\VISTAVIRTUAL';
```

Tentoraz sa vypíšu len údaje pre konkrétneho používateľa.

Výpis parametrov a vlastností

Konzolová aplikácia SQL Server Management Studio umožňuje aj výpis parametrov a to na rôznej úrovni. Pomocou klauzuly SERVERPROPERTY môžeme zisťovať niektoré parametre databázového servera, napríklad:

```
SELECT SERVERPROPERTY('ServerName')
SELECT SERVERPROPERTY('Edition')
SELECT SERVERPROPERTY('ProductVersion')
SELECT SERVERPROPERTY('ProductLevel')
```

Server nám vráti hodnoty požadovaných parametrov v tvare:

```
VISTAVIRTUAL_PC
Enterprise Evaluation Edition
10.0.1600.22
RTM
```

Pomocou klauzuly DATABASEPROPERTY môžeme zisťovať parametre konkrétnej databázy, napríklad:

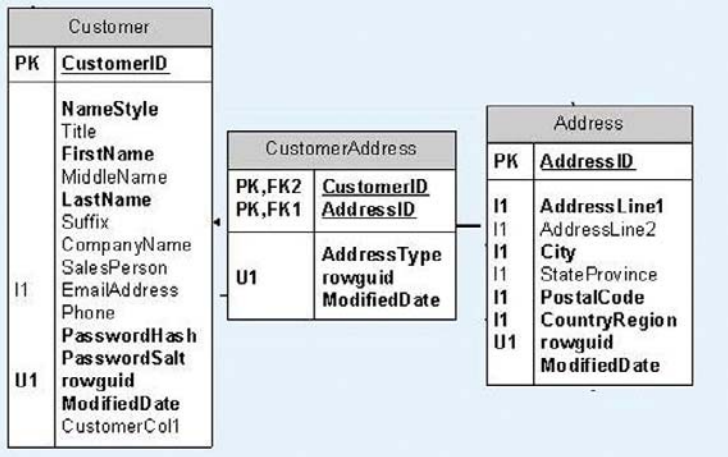
```
SELECT DATABASEPROPERTYEX('Pokusy', 'Status')
SELECT DATABASEPROPERTYEX('Pokusy', 'Recovery')
SELECT DATABASEPROPERTYEX('Pokusy', 'Collation')
SELECT DATABASEPROPERTYEX('Pokusy', 'Updateability')
SELECT DATABASEPROPERTYEX('Pokusy', 'UserAccess')
SELECT DATABASEPROPERTYEX('Pokusy', 'IsAutoCreateStatistics')
SELECT DATABASEPROPERTYEX('Pokusy', 'IsAutoShrink')
```

Server nám vráti hodnoty v tvare:

```
ONLINE
FULL
Slovak_CI_AS
READ_WRITE
MULTI_USER
1
0
```

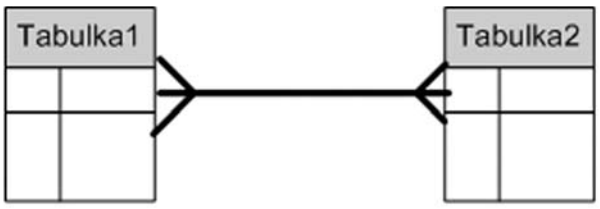
V kontextovom menu okna pre zadávanie dotazov „Design Query Editor" môžeme aktivovať nástroj pre grafický interaktívny návrh SQL dopytu. Pre ilustráciu možností tohto nástroja vytvoríme dopyt nad troma relačne zviazanými tabuľkami

- Customer
- CustomerAddress
- Addres

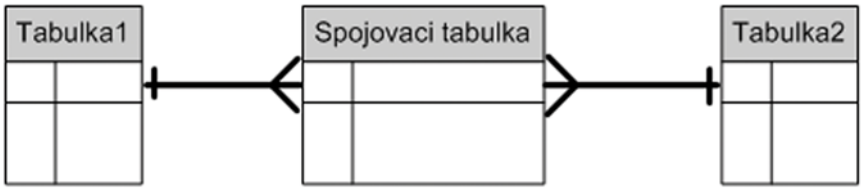


Tabuľky databázy

Aby sme rozumeli väzbám medzi týmito tabuľkami ukážeme princíp dekompozície dvoch tabuliek na relačnú väzbu pomocou spojovacej tabuľky. Väčšina databázových systémov totiž nedokáže priamo pracovať zo vzťahmi typu N:M, v praxi sa používa dekompozícia, to znamená, že tento vzťah sa implementuje pomocou spojovacej tabuľky. To znamená, že vzťah N:M môžeme rozložiť na dva vzťahy typu N:1

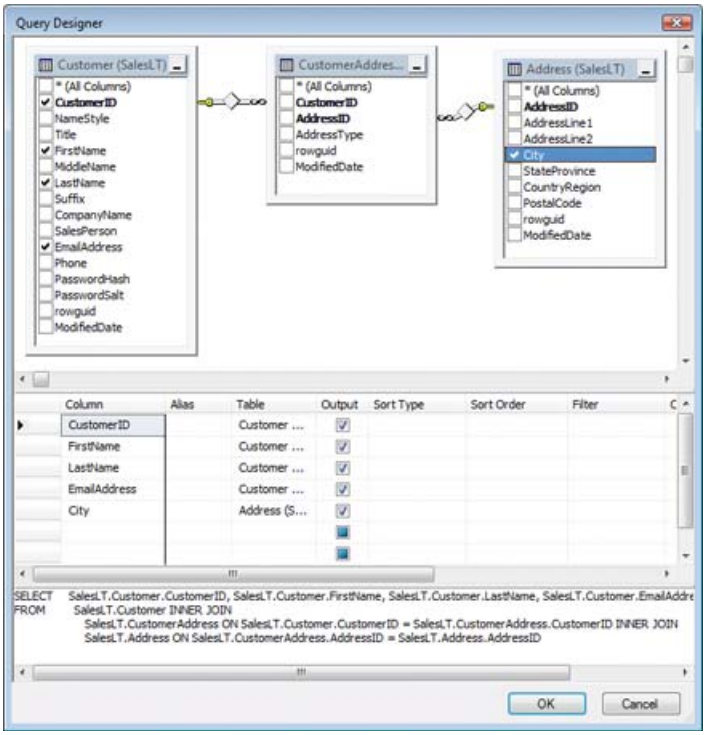


Vzťah „viaceré k viacerým"



Rozloženie vzťahu „viaceré k viacerým" pomocou spojovacej tabuľky

Po umiestnení tabuliek na plochu sa automaticky zahrnú relačné väzby, ktoré boli medzi nimi definované



Query Editor

Takto sme na pár kliknutí navrhli dotaz to troch relačne zviazaných tabuliek

```
SELECT  SalesLT.Customer.CustomerID, SalesLT.Customer.FirstName,
        SalesLT.Customer.LastName, SalesLT.Customer.EmailAddress,
        SalesLT.Address.City
FROM    SalesLT.Customer INNER JOIN
        SalesLT.CustomerAddress ON SalesLT.Customer.CustomerID =
        SalesLT.CustomerAddress.CustomerID INNER JOIN
        SalesLT.Address ON SalesLT.CustomerAddress.AddressID =
        SalesLT.Address.AddressID
```

Môžeme si ho overiť spustením

```
CustomerID  FirstName    LastName     EmailAddress                          City
----------- ------------------------------------------------------------------
29485       Catherine    Abel         catherine0@adventure-works.com        Van Nuys
29486       Kim          Abercrombie  kim2@adventure-works.com              Branch
29489       Frances      Adams        frances0@adventure-works.com          Modesto
29490       Margaret     Smith        margaret0@adventure-works.com         Lewiston
```

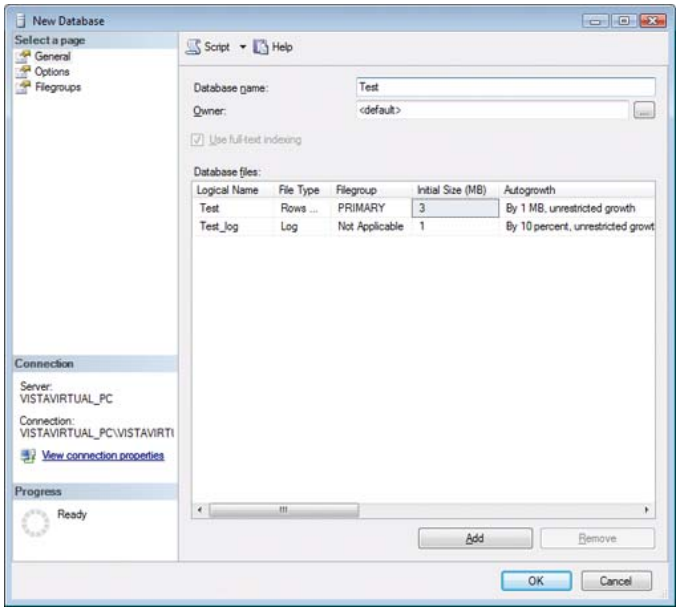
Vytvorenie novej databázy pre štúdium a pokusy

Pre naše pokusy s databázovým serverom je vhodné vytvoriť novú databázu. Môžeme ju vytvoriť príkazom:

CREATE DATABASE názov_databázy

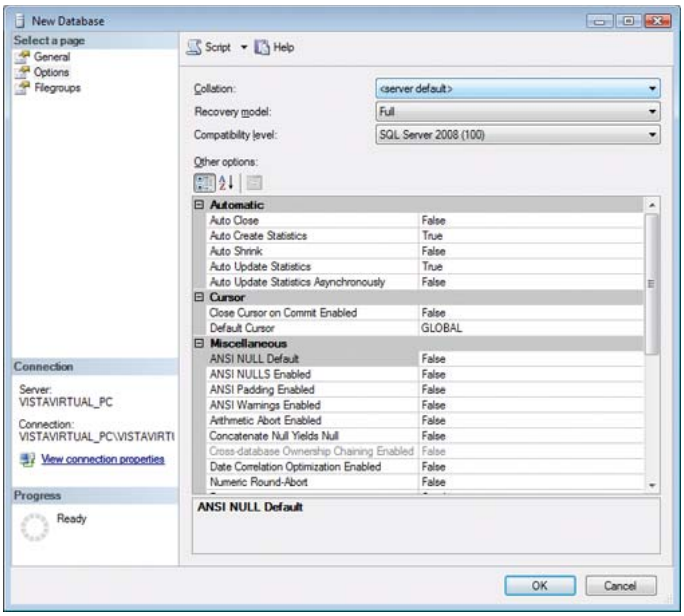
Alebo pomocou administrátorského nástroja SQL Server Management Studio v kontextovom menu Databases – New Database. Všimnite si, že toto dialógové okno má tri stránky:

- General
- Options
- Filegroups



Dialóg pre vytvorenie novej databázy – záložka General

V záložke Options najčastejšie nastavujeme kódovú stránku pre vyhľadávanie a triedenie. Jazyk SQL je dosť blízky prirodzenému jazyku (angličtine), aj dopytovanie a prípadné utrieďovanie výsledkov vyhľadávania by malo rešpektovať zvyklosti prirodzeného jazyka. Po príklad nemusíme chodiť ďaleko. Poslúži nám typická česko-slovenská hláska CH. Parametre pre porovnávanie a triedenie sa nastavujú pomocou parametra COLLATE napríklad pri vytváraní tabuliek. Parameter COLLATE, môžeme nastaviť buď pre celú tabuľku, alebo aj jednotlivo pre niektoré stĺpce.



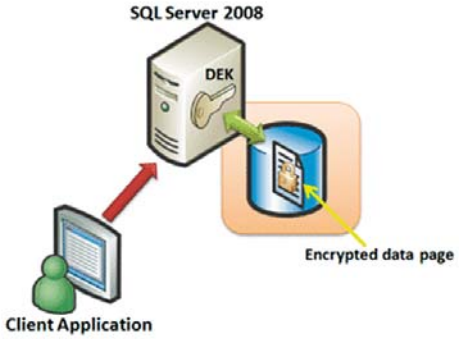
Záložka Options pre nastavenie parametrov

Kapitola 3: Stručný prehľad noviniek servera SQL Server 2008

Nová verzia databázového servera Microsoft SQL Server 2008 je svojimi vlastnosťami, robustnosťou a spoľahlivosťou predurčená aj pre aplikácie typu „mission-critical", pričom zároveň znižuje nároky na infraštruktúru a jej správu. Ak by sme medzi sebou porovnávali dve predchádzajúce verzie 2000 a 2005, dalo by sa bez zveličovania povedať, že SQL Server 2005 predstavoval revolučnú inováciu, hlavne v oblasti podpory XML ako natívneho formátu a filozofie univerzálneho dimenzionálneho modelovania (UDM) pre BI projekty. Významné novinky prináša aj verzia 2008, aj keď zmeny v porovnaní s verziou 2005 už nie sú až také prevratné. Veľmi zaujímavou črtou je TABLIX. TABLIX je akronymom zložený z častí slov TABLe a matrIX. Spája výhody bežnej a kontingenčnej tabuľky (matice hodnôt). Reporty sú v novej verzii koncipované tak, aby poskytovali operatívny prehľad nielen aktuálnych hodnôt, ale aj ich trendov. V aktuálnom CTP je nástroj Report Designer pre návrh reportov rozšírený o nové typy grafickej prezentácie relačných a multidimenzionálnych údajov. Na serveri je možné pridať nielen pamäť, ale aj dodatočnú procesorovú kapacitu (Hot Add CPU) bez nutnosti vypnutia databázového servera.

Bezpečnosť
Už pri letmom pomyslení na možný výpadok databázového servera, alebo nebodaj stratu údajov naskakujú databázovým administrátorom na chrbte zimomriavky. A niet sa ani čomu diviť. Odborníci spočítali, že jedna minúta výpadku databázového servera príde niekedy aj na 10 000 USD. Preto je snaha, aby sa takéto situácie nevyskytovali.

Pri analýze príčin výpadku databázových serverov sa dospelo zhruba k nasledovnému rozdeleniu:
– 3 percentá tvoria katastrofy (netreba vysvetľovať, 11. september 2001 hovorí za všetko),
– 36 percent výpadkov je zapríčinených chybami obsluhy,
– 46 percent príčin výpadkov tvoria poruchy hardvéru a podobne.

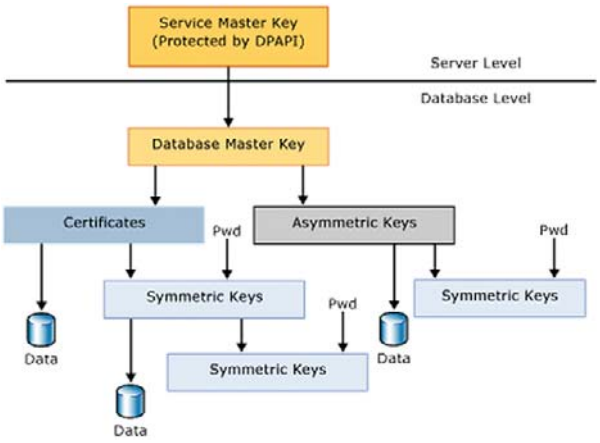


Transparentné šifrovanie údajov

Ukradnúť databázu bolo dosiaľ pomerne jednoduché. Pri aktuálnych verziách SQL servera stačilo odpojiť databázu, alebo len zastaviť databázový server prekopírovať „mdf" a „ldf" súbory na svoj počítač, pripojiť databázu k svojej inštancii SQL servera a... údaje sú naše.

SQL Server 2008 umožňuje zvýšiť bezpečnosť databázových aplikácií transparentným **šifrovaním** databáz, údajových a logovacích súborov. Toto umožňuje získať organizáciám požadované certifikáty napríklad pre správu osobných údajov, prípadne naplniť prísne kritériá a predpisy informačnej bezpečnosti. Šifrovanie údajov je možné implementovať bez nutnosti akýchkoľvek zmien v súvisiacich aplikáciách. So šifrovaním úzko súvisí aj **správa kľúčov**. SQL Server 2008 podporuje aj hardvérové bezpečnostné moduly pre správu kľúčov a autentifikáciu od rôznych firiem. Konsolidácia správy kľúčov podstatne zjednodušuje správu šifrovania údajov vo veľkých viacserverových údajových centrách. Pri zabezpečení databázového servera ako integrálnej súčasti informačného systému je dôležitá aj možnosť auditovania. SQL Server 2008 umožňuje pokročilé **auditovanie** prístupu do databázy, dopytovania a modifikovania údajov. Tento nástroj spoľahlivo dokumentuje ktoré údaje, kedy a kým boli čítané, prípadne modifikované.

SQL Server 2008 podporuje symetrické a asymetrické šifrovacie kľúče a digitálne certifikáty. Hierarchia kľúčov je názorne ukázaná na obrázku.

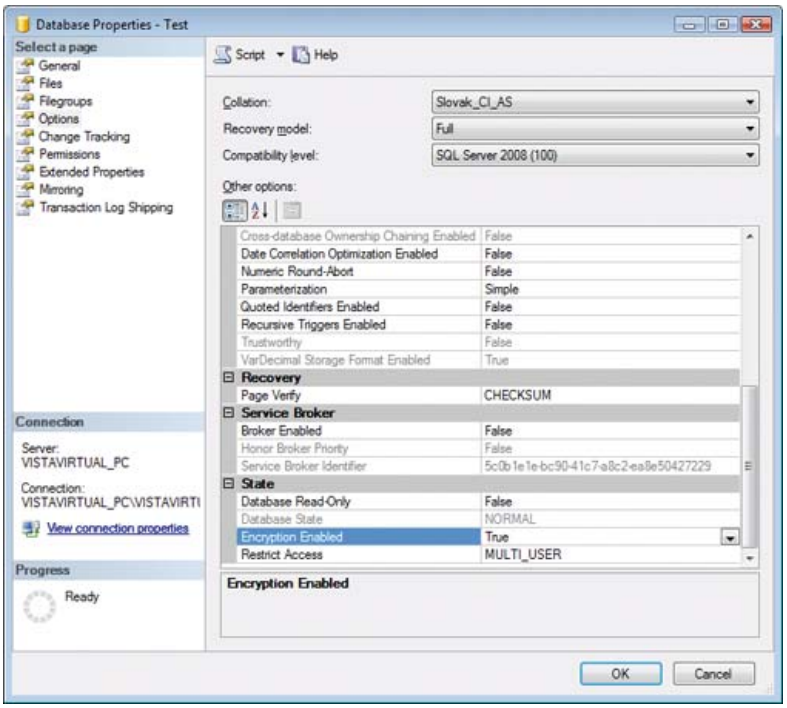


Správa kľúčov

Aktivovaním črty Transparent Data Encryption šifrovania na databáze bude zašifrovaný údajový súbor aj transakčný log na disku. Nezašifrované sú len údaje v cache pamäte. Transparent Data Encryption v princípe šifruje/dešifruje v reálnom čase diskové I/O operácie. Pre aplikácie pristupujúce k údajom sa nič nemení, prístup k údajom je riadený „grantovaním" práv. Pre šifrovanie databázy slúži špeciálny kľúč „database encryption key" (DEK), ktorý je uložený v „boot" zázname databázy pre dostupnosť údajov aj počas „recovery". DEK je chránený certifikátom uloženým v „master" databáze. Údaje sú šifrované použitím algoritmov AES128, AES192, AES256 alebo TripleDES. („Default" je AES128.)

Šifrovanie môžeme povoliť už pri vytváraní novej databázy nastavením parametra „Encryption Enabled" na hodnotu „True".

Povolenie šifrovania v dialógu pre nastavovanie vlastností databázy



Pre vytvorenie MASTER KEY slúži príkaz:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'nbusr123'
```

Nový certifikát vytvoríme príkazom:

```
CREATE CERTIFICATE NasCertifikat
       ENCRYPTION BY PASSWORD = 'nbusr123'
       WITH SUBJECT = 'Code signing certificate'
GO
```

Pre certifikát vytvoríme používateľa:

```
CREATE USER certUser FOR CERTIFICATE NasCertifikat
```

Pridelíme mu privilégiá, napríklad pre výber údajov z tabuľky:

```
GRANT SELECT ON zakaznici TO certUser
GO
```

Vytvoríme šifrovací kľúč:

```
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE NasCertifikat
```

a zapneme šifrovanie pre databázu, na ktorú chceme šifrovanie uplatniť.

```
ALTER DATABASE MojaDB SET ENCRYPTION ON
```

Stav šifrovania sa dá overiť pohľadom sys.dm_database_encryption_keys.

Ak chceme využiť certifikát pre uloženú procedúru (použijeme príklad z inej kapitoly):

```
CREATE PROCEDURE MultiZadavanie(@id int, @meno nvarchar(30),
@email nvarchar(30))
AS
BEGIN
INSERT INTO dbo.zamestnanci
values(@id, @meno, @email)
END
```

Pridáme ho príkazom:

```
ADD SIGNATURE TO dbo.MultiZadavanie BY CERTIFICATE NasCertifikat
       WITH PASSWORD = 'nbusr123'
```

Kompresia údajov a záloh
SQL Server 2008 umožňuje kompresiu údajov v databázových tabuľkách a zálohách. Ušetrí sa nielen miesto na disku ale paradoxne aj čas, nakoľko kompresia údajov v operačnej pamäti je podstatne rýchlejšia ako operácie zápisu na disk, pretože disk nech by bol akýkoľvek výkonný, je na rozdiel od „elektronickej" pamäte mechanické pamäťové médium. Kompresia údajov v databázových tabuľkách môže prebiehať buď na úrovni záznamov, teda riadkov tabuľky, alebo na úrovni stránky.

Správa zdrojov (Resource Governor)
SQL Server 2008 obsahuje službu Resource Governor, kde je možné nastaviť hranice využívania jednotlivých zdrojov. **Resource Governor je popísaný podrobne vo štvrtej kapitole**.

Vynútenie politiky cez Declarative Management Framework
Declarative Management Framework je nový framework, ktorý umožňuje centralizovanú správu inštancií servera SQL Server 2008 v oblasti dodržiavania politík pre zvolenú systémovú konfiguráciu. Definované pravidlá platia pre jednotlivé inštancie alebo objekty v inštancii ako sú databázy, tabuľky, pohľady, uložené procedúry. Monitoruje zmeny a dokáže zabrániť neautorizovaným zásahom dôslednou kontrolou dodržovania politík pre požadovanú konfiguráciu. Declarative Management Framework podstatne znižuje náklady na správu, nakoľko zjednodušuje administrátorské úlohy. Umožňuje explicitnú a automatizovanú administráciu. **Declarative Management Framework je popísaný podrobne vo štvrtej kapitole**.

Prediktívny systém optimalizácie výkonu
Požiadavky na výkon databázového servera rôzne kolíšu. Sú rôzne pre rôzne oddelenia a rôzne obdobia kalendárneho, alebo častejšie fiškálneho roka. Optimalizácia výkonu pri starších verziách bola činnosť značne časovo náročná a keďže ju vykonávajú vysoko kvalifikovaní administrátori tak aj drahá. SQL Server 2008 obsahuje množinu čŕt pre prediktívnu a škálovateľnú optimalizáciu výkonu vrátane centrálneho repozitára pre ukladanie optimalizovaných údajov a reportovacích a monitorovacích nástrojov.

Rozšírená správa udalostí
SQL Server 2008 obsahuje črtu nazývanú „SQL Server Extended Events" pre správu udalostí. Tento mechanizmus zachytáva, filtruje a triedi udalosti generované serverovými procesmi. To umožňuje rýchlu a efektívnu diagnostiku problémov za behu servera, monitorovanie zásobníkov volaní procedúr procedurálneho jazyka T-SQL.

Azda najčastejšie sa stretneme s novými črtami v databázovom jazyku SQL. Tieto novinky je totiž možné využívať v relačných databázach, v dopytoch pre vytvorenie reportov a podobne. Preto predstavíme najvýznamnejšie novinky podrobnejšie.

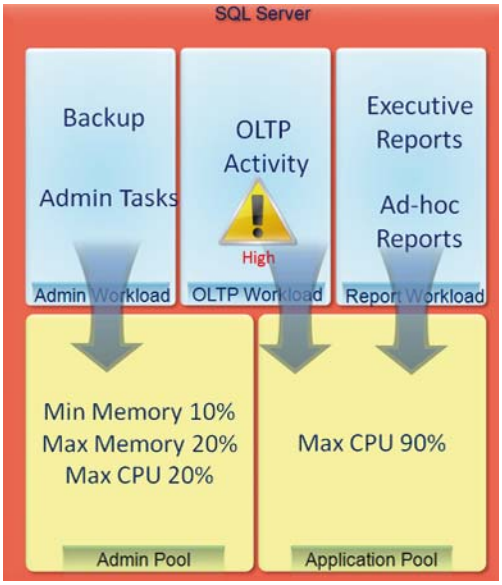
Kapitola 4: Správa zdrojov (Resource Governor)

Záťaž databázových serverov počas fiškálneho roka značne kolíše a dimenzovanie ich výkonu a nastavenie zdieľania zdrojov zďaleka nie je jednoduchá úloha. Napríklad v obdobiach zvýšenej aktivity zákazníkov (Vianoce...) sú servery, na ktorých sa spracovávajú objednávky zaťažené naplno. Podobne je to so servermi na ekonomickom oddelení v období, kedy sa spracovávajú koncoročné uzávierky, výkazy, vyhodnotenia, podklady pre daňové doklady a podobne. Nezriedka sa uvažuje o inovácii výpočtovej kapacity. Databázový server je komplexná platforma pre rôzne služby, počnúc klasickými transakciami, zálohami údajov, analytickými službami, reportovaním, pričom v rôznych oblastiach nasadenia a v rôznych aplikáciách majú rôzne služby rôzne nároky na zdroje, pričom kritickými zdrojmi sú hlavne výpočtová kapacita procesorov a pamäť. SQL Server 2008 obsahuje službu Resource Governor, kde je možné nastaviť hranice využívania jednotlivých zdrojov.

Ak niekto vytvorí dopyt, alebo report, ktorý spotrebováva veľa zdrojov, napríklad procesorového času, môže tieto zdroje alokovať pomocou príkazov:

```
ALTER RESOURCE POOL WITH(MAX_CPU_PERCENT = 20)
ALTER RESOURCE POOL WITH(MAX_CPU_PERCENT = 80)
```

po takomto nastavení dve úlohy už nesúperia o zdroje, nakoľko ich prerozdelenie bolo explicitne dané.



Nastavenie alokácie zdrojov pre jednotlivé služby a procesy

Pre definovanie záťaže je potrebné definovať objekt WORKLOAD GROUP

```
CREATE|ALTER|DROP WORKLOAD GROUP
{group_name | "default"}
[WITH <Policy specification>]
[USING {pool_name | "default"}]
ALTER RESOURCE POOL WITH(MAX_CPU_PERCENT = 80)
```

napríklad:

```
CREATE WORKLOAD GROUP newReports USING „default"
```

A nastaviť preň parametre:

```
IMPORTANCE = {LOW | MEDIUM | HIGH}
REQUEST_MAX_MEMORY_GRANT_PERCENT = value
```

```
REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value
REQUEST_MAX_CPU_TIME_SEC = value
MAX_DOP = value
GROUP_MAX_REQUESTS = value
```

Následne vytvoríme Resource pool:

```
CREATE|ALTER|DROP RESOURCE POOL
{pool_name | default}
WITH
       ([MIN_CPU_PERCENT = value]
       [[,]MAX_CPU_PERCENT = value]
       [[,]MIN_MEMORY_PERCENT = value]
       [[,]MAX_MEMORY_PERCENT = value]) [;]
```

napríklad:

```
CREATE RESOURCE POOL bigPool
GO
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

a záťaž:

```
ALTER RESOURCE POOL „default"
WITH
     ( MAX_CPU_PERCENT = 25)
GO
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

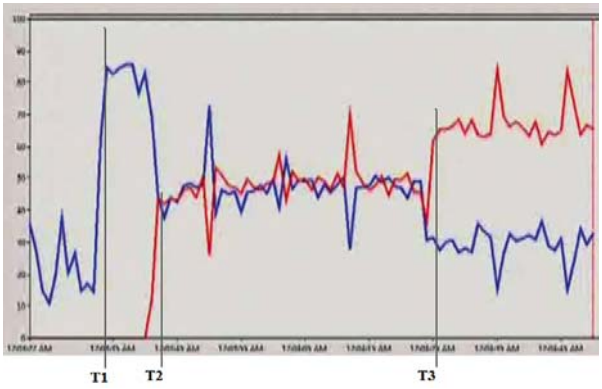
Uvedieme komplexnejší príklad:

```
CREATE RESOURCE POOL poolLimited
WITH
(
   MAX_CPU_PERCENT=30,
   MAX_MEMORY_PERCENT=50


CREATE WORKLOAD GROUP wrkgroupLimited
WITH
(
    IMPORTANCE = MEDIUM ,
    REQUEST_MAX_MEMORY_GRANT_PERCENT = 50,
    REQUEST_MAX_CPU_TIME_SEC = 300,
    REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 300,
    MAX_DOP = 8,
    GROUP_MAX_REQUESTS = 30

USING poolLimited;
```

Pre získanie prehľadu o prerozdelení záťaže slúžia pohľady:

sys.dm_resource_governor_workload_groups
sys.dm_resource_governor_resource_pools
sys.dm_resource_governor_configuration

Črtu Resource Governor môžeme otestovať simulovaním záťaže dvoch rovnakých procesov. V okamihu T1 (viď obrázok) bol spustený prvý záťažovo náročný proces. Až do okamihu T2, spotrebovával dostupné zdroje prakticky sám. V okamihu T2 bol spustený druhý rovnako náročný proces. Od tohto okamihu procesy súperia o zdroje, pričom vidíme, že záťaž sa medzi obidva procesy rozdelila približne rovnako. V okamihu T3 sme nastavili pre proces 1 20 percent zdrojov a pre proces 2 80 percent. Prerozdelenie zdrojov od tohto okamihu vidíme názorne na obrázku.



Sledovanie záťaže dvoch procesov pred a po nastavení záťaže

Nakoniec ukážeme príklad nastavenia záťaže pre tri skupiny používateľov. Príslušnosť používateľa do skupiny sa identifikuje automaticky, podľa toho z akej aplikácie používateľ k serveru SQL Server pristupuje. Prvá skupina bežných používateľov využíva Query Analyser, administrátori pristupujú cez SQL Server Management Studio a pre generovanie reportov sa využíva report server.

```
BEGIN TRAN
CREATE WORKLOAD GROUP groupAdhoc
CREATE WORKLOAD GROUP groupReports
CREATE WORKLOAD GROUP groupAdmin
GO

CREATE FUNCTION rgclassifier_v1() RETURNS SYSNAME
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @grp_name AS SYSNAME
      IF (SUSER_NAME() = 'sa')
        SET @grp_name = 'groupAdmin'
      IF (APP_NAME() LIKE '%MANAGEMENT STUDIO%')
          OR (APP_NAME() LIKE '%QUERY ANALYZER%')
          SET @grp_name = 'groupAdhoc'
      IF (APP_NAME() LIKE '%REPORT SERVER%')
          SET @grp_name = 'groupReports'
    RETURN @grp_name
END
GO

ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION= dbo.rgclassifier_v1)
COMMIT TRAN
GO
-- Start Resource Governor
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

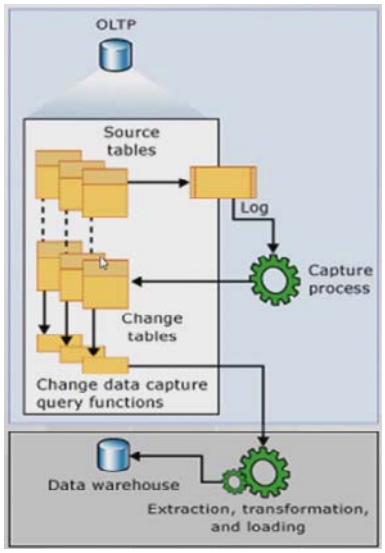
Pre každú skupinu nastavíme percentuálne rozdelenie záťaže. Napríklad pre administrátorov:

```
BEGIN TRAN
-- Create a new resource pool and set resource limits.
CREATE RESOURCE POOL poolAdmin
WITH (
    MIN_CPU_PERCENT = 10,
    MIN_MEMORY_PERCENT = 10,
    MAX_MEMORY_PERCENT = 10
)
;
ALTER WORKLOAD GROUP groupAdmin
USING poolAdmin;
COMMIT TRAN
GO
-- Apply the changes to the Resource Governor in-memory configuration.
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

Kapitola 5: Zachytávanie zmien v databázovej tabuľke

Táto nová črta (anglicky CDC – Change data capture) nám umožňuje podchytiť zmeny v databázovej tabuľke, napríklad pridanie alebo vymazanie záznamov, prípadne ich zmeny bez toho, aby sme museli pre tento účel použiť spúšte (triggers).

Keď sa zmenia údaje v zdrojovej tabuľke, tieto údaje sú zapísané do transakčných logov. Proces monitorovania zmien pracuje práve s týmto transakčným logom a zapisuje údaje do tabuliek zmien. Tieto údaje sú potom k dispozícii napríklad pre proces ETL (Extrakcia, Transformácia, Loading), teda pre zavádzanie údajov do údajových skladov.



Princíp zachytávania zmien v databázových tabuľkách

Najlepšie bude predstaviť túto novinku na praktickom príklade. Pre tento príklad si vytvoríme samostatnú databázu. Môžeme to urobiť v nástroji SQL Server Management Studio pomocou kontextového menu na priečinku „Databases" alebo cez klientsku konzolovú aplikáciu (aktivuje sa tlačidlom „New Query") príkazom

```
CREATE DATABASE CDC_databaza
```

Konzolovú aplikáciu prepneme do novej databázy:

```
USE CDC_databaza
```

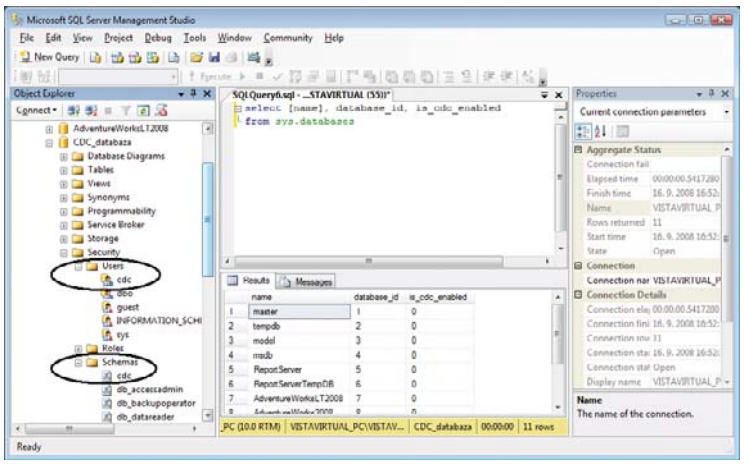
Režim zachytávania zmien v aktuálnej databáze aktivujeme spustením uloženej procedúry:

```
EXEC sp_cdc_enable_db
```

Pripomíname, že pre spustenie tejto uloženej procedúry musíme mať privilégiá administrátora. Ak chceme zistiť, či je pre existujúce databázy táto črta povolená, môžeme vytvoriť dopyt do katalógového pohľadu sys. databases, kde nás zaujíma názov databázy, jej ID a príznak is_cdc_enabled. Ak je tento príznak nastavený na hodnotu 1, zachytávanie zmien je povolené.

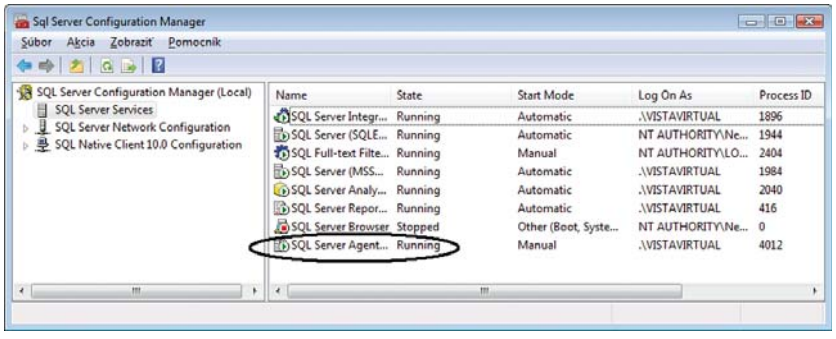
```
select [name], database_id, is_cdc_enabled from sys.databases

name                           database_id is_cdc_enabled
------------------------------ ----------- --------------
master                         1           0
tempdb                         2           0
model                          3           0
msdb                           4           0
ReportServer                   5           0
ReportServerTempDB             6           0
AdventureWorksLT2008           7           0
AdventureWorks2008             8           0
Pokusy                         9           0
Test                           10          0
CDC_databaza                   11          1
```

V nástroji SQL Server Management Studio sa môžeme presvedčiť, že zavolaním uloženej procedúry pre aktiváciu CDC pribudol do zoznamu používateľov v priečinku „Security" nový používateľ CDC a v záložke „Schemas" rovnomenná schéma.



Do zoznamu používateľov v priečinku „Security" pribudol nový používateľ CDC a v záložke „Schemas" rovnomenná schéma

Pre fungovanie CDC je potrebné spustiť službu SQL Server Agent. V implicitnej inštalácii je totiž zastavená s príznakom, že sa spúšťa ručne. Môžeme ju spustiť napríklad prostredníctvom nástroja Sql Server Configuration Manager.



Spustenie služby SQL Server Agent

Pre pokusy zo zachytávaním zmien vytvoríme jednoduchú tabuľku zamestnancov. Ak chceme nastaviť parameter supports_net_changes na hodnotu 1 je potrebné, aby tabuľka mala definovaný primárny kľúč:

```
CREATE TABLE zamestnanci
(
   ID int PRIMARY KEY,
   meno nvarchar(30),
   email nvarchar(30))
```

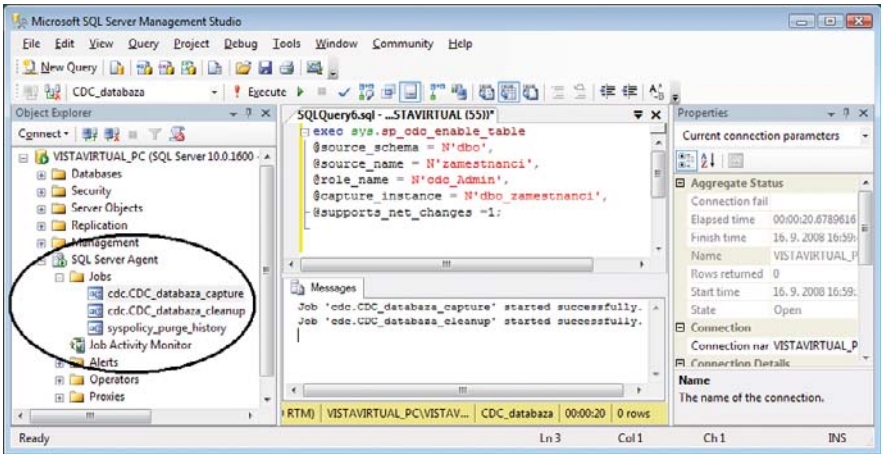
CDC je potrebné povoliť a nastaviť separátne pre každú tabuľku pomocou uloženej procedúry:

```
exec sys.sp_cdc_enable_table
@source_schema = N'dbo',
@source_name = N'zamestnanci',
@role_name = N'cdc_Admin',
@capture_instance = N'dbo_zamestnanci',
@supports_net_changes =1;
```

Po spustení tejto uloženej procedúry sa vytvoria dve úlohy pre SQL Server Agent. Jedna pre zachytávanie zmien z transakčného logu a pre ich synchrónny zápis do asociovaných tabuliek určených pre zachytávanie zmien a druhá pre čistenie.

```
Job 'cdc.CDC_databaza_capture' started successfully.
Job 'cdc.CDC_databaza_cleanup' started successfully.
```

Môžeme sa o tom presvedčiť aj prostredníctvom nástroja SQL server Management Studio



Vytvorenie úloh pre SQL Server Agent

Ak chceme zistiť, pre ktoré tabuľky je povolené zachytávanie zmien, môžeme vytvoriť dopyt do katalógového pohľadu sys.tables, kde nás zaujíma názov tabuľky a príznak is_tracked_by_cdc. Ak je tento príznak nastavený na hodnotu 1, zachytávanie zmien je povolené.

```
select [name], is_tracked_by_cdc from sys.tables

name                 is_tracked_by_cdc
-------------------- -----------------
ddl_history          0
lsn_time_mapping     0
captured_columns     0
index_columns        0
zamestnanci          1
dbo_zamestnanci_CT   0
systranschemas       0
change_tables        0
```

Do tabuľky pridáme niekoľko záznamov:

```
INSERT INTO dbo.zamestnanci VALUES (1, 'Mickey Mouse','mickey@disney.com');
INSERT INTO dbo.zamestnanci VALUES (2, 'Donald Duck','donald@disney.com');
INSERT INTO dbo.zamestnanci VALUES (3, 'Snehurka','snehurka@disney.com');
```

Najskôr si vypíšeme obsah tabuľky v ktorej sme vykonávali zmeny, čiže tabuľku zamestnancov, do ktorej sme pridali tri záznamy:

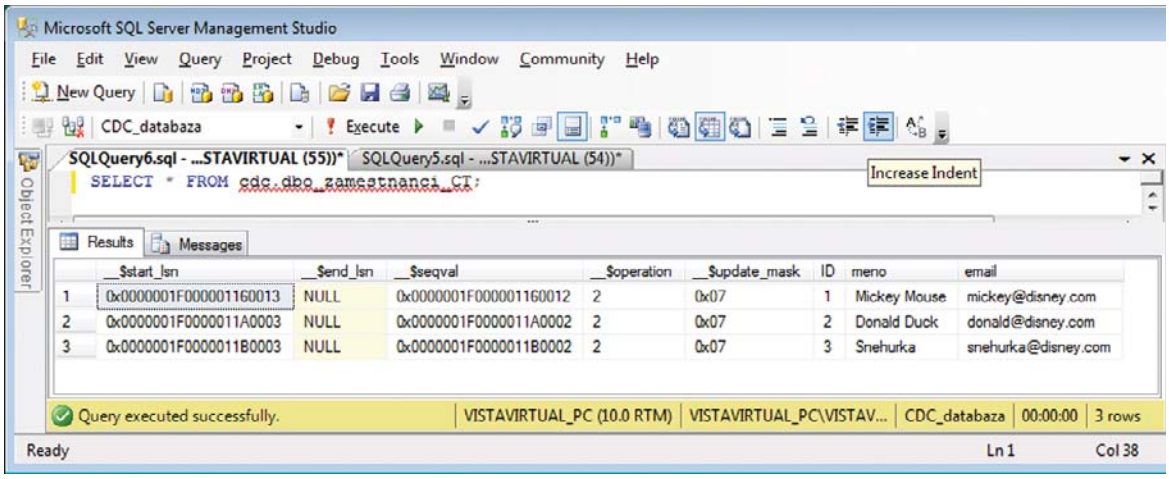
```
SELECT * FROM dbo.zamestnanci

ID          meno                           email
----------- ------------------------------ ------------------------------
1           Mickey Mouse                   mickey@disney.com
2           Donald Duck                    donald@disney.com
3           Snehurka                       snehurka@disney.com
```

Vypíšeme obsah tabuľky pre zachytávanie zmien:

```
SELECT * FROM cdc.dbo_zamestnanci_CT;
```

Výpis je z dôvodu tlače rozdelený na dve časti.

```
__$start_lsn            __$end_lsn     __$seqval
----------------------- -------------- -----------------------
0x0000001C000000480013 NULL           0x0000001C000000480012
0x0000001C0000004C0003 NULL           0x0000001C0000004C0002
0x0000001C0000004D0003 NULL           0x0000001C0000004D0002


__$operation __$update_mask   ID    meno            email
------------ ---------------- ---- --------------- --------------------
2            0x07             1    Mickey Mouse    mickey@disney.com
2            0x07             2    Donald Duck     donald@disney.com
2            0x07             3    Snehurka        snehurka@disney.com
```



Výpis tabuliek zmien

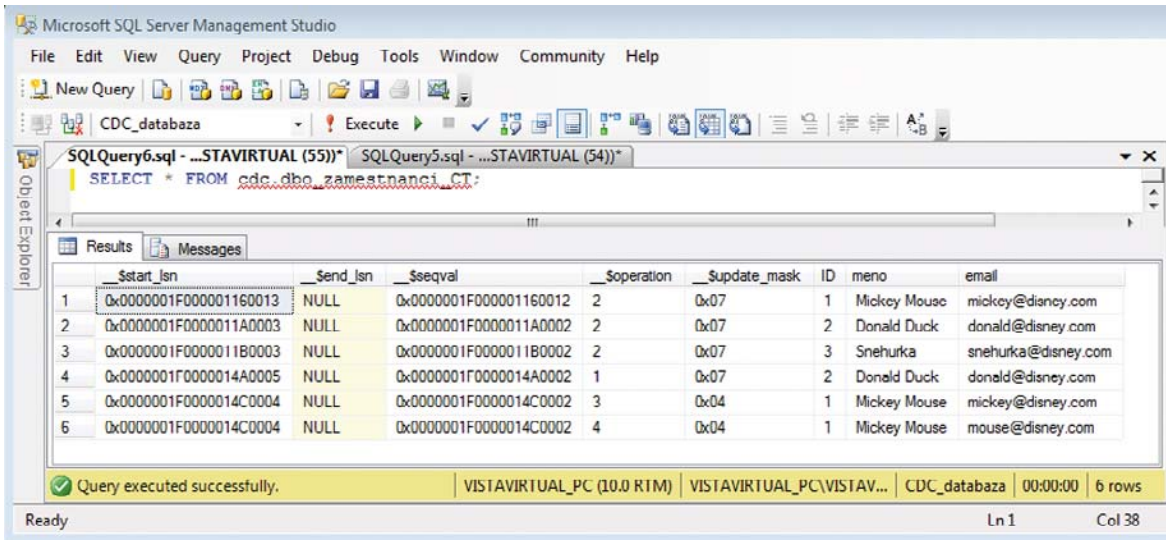
Atribúty `__$start_lsn a __$end_lsn` udávajú polohu zachytenej zmeny v redo-log súbore. Atribút `__$operation` udáva typ operácie, ktorá bola nad zachyteným záznamom vykonaná. Hodnota 1 znamená operáciu DELETE, hodnota 2 operáciu INSERT a 3 je pre UPDATE.

Aby ste sa presvedčili o fungovaní zachytávania zmien, urobte ešte ďalšie dve operácie. Jeden záznam vymažte a jeden zmeňte:

```
DELETE FROM dbo.zamestnanci WHERE ID=2
UPDATE dbo.zamestnanci SET email = 'mouse@disney.com' WHERE ID=1
```

Následne si pozrite aké údaje obsahuje tabuľka zamestnancov a čo sa zapísalo do tabuľky zachytávania zmien. Jej obsah môžete vypísať pomocou SQL príkazu:

```
SELECT * FROM cdc.dbo_zamestnanci_CT;
```



Výpis tabuliek po vykonaných zmenách

Kapitola 6: Vynútenie dodržiavania politík cez Declarative Management Framework

Declarative Management Framework je nový framework, ktorý umožňuje centralizovanú správu inštancií SQL Serveru 2008 v oblasti dodržiavania politík pre zvolenú systémovú konfiguráciu. Definované pravidlá platia pre jednotlivé inštancie alebo objekty v inštancii ako sú databázy, tabuľky, pohľady, uložené procedúry. Monitoruje zmeny a dokáže zabrániť neautorizovaným zásahom dôslednou kontrolou dodržovania politík pre požadovanú konfiguráciu. Declarative Management Framework podstatne znižuje náklady na správu, nakoľko zjednodušuje administrátorské úlohy. Umožňuje explicitnú a automatizovanú administráciu.

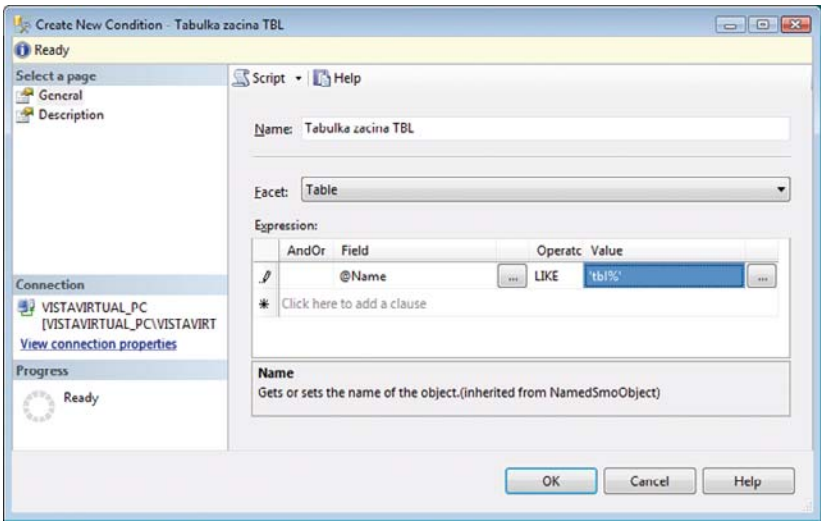
Môžeme napríklad definovať politiku pre názvy objektov. Niekedy to vývojárov občas zvádza k používaniu krátkych často jednopísmenových názvov premenných a atribútov, napríklad A, B1.

Z dôvodu prehľadnosti kódu nie je príliš vhodný ani druhý extrém – rozsiahle a vyčerpávajúce názvy napríklad. Tabulka_Intervalu_olejovania_osi_kluky_spinaca_vytahoveho_stykaca

Veľmi dobrý a prehľadný je spôsob označovania premenných nazývaný Maďarská notácia (zaviedol ju jeden programátor Microsoftu maďarského pôvodu), kde okrem jedno, prípadne dvojslovného názvu objektu alebo premennej, naznačíme pomocou prvého písmena, prípadne prvých troch písmen aj typ objektu alebo j dátový typ atribútu alebo premennej. Aby sme ukázali ako to funguje, definujeme podmienku, že názvy všetkých novovytváraných tabulliek sa musia začínať prefixom „tbl". Potom nebudeme môcť vytvoriť tabuľku „zamestnanci" ale v zmysle definovanej politiky len „tblZamestnanci"

Pre praktický príklad použitia spustíme nástroj SQL Server Management Studio a pripojíme sa k databázovému serveru. V okne „Object Explorer " rozvinieme postupne položky „Management", "Policy Management" a „Facets", Pravým tlačidlom myši klikneme na „Table" následne na položku „New Condition". Podmienku vhodne pomenujeme a zo zoznamu vyberieme typ objektu ktorého sa podmienka bude týkať.Následne v okne „Expressioin" Zostavíme podmienku napríklad s názvom „Tabulka začína tbl". Podmienku bude tvoriť výraz.

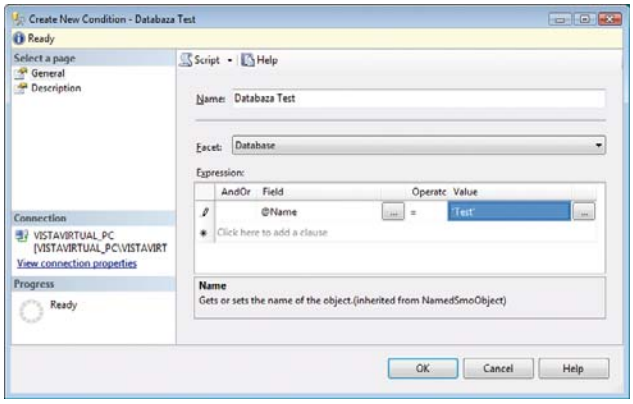
```
@Name LIKE 'tbl%'
```



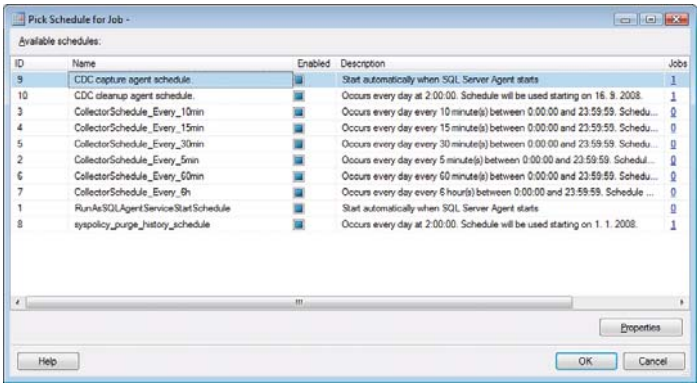
Nová podmienka

V zložke „Policy“ pomocou kontextovéno menu „New Policy“ definujeme príslušnú politiku. V dialógu „Create New Policy“ politiku pomenujeme, v našom príklade napríklad „Konvencia nazvov“.

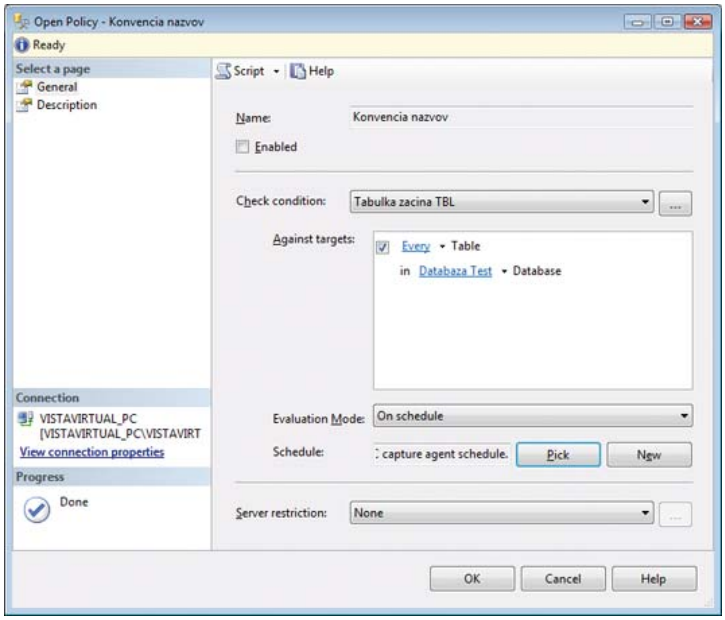
Mód spúšťania môžeme nastaviť „On Demand“ alebo „On Schedule“. V našom príklade nastavíme mód „On Schedule“, vyberieme časový plán a politiku obmedzíme len na databázu Pokusy.



Nová podmienka pre výber databázy „Pokusy“



Výber Schedule Job



Definovanie politiky

Vytvorme novú tabuľku v databáze „Pokusy“:

```
CREATE TABLE Zamestnanci
(
   ID int NOT NULL,
   meno nvarchar(30),
   email nvarchar(30)
)
```

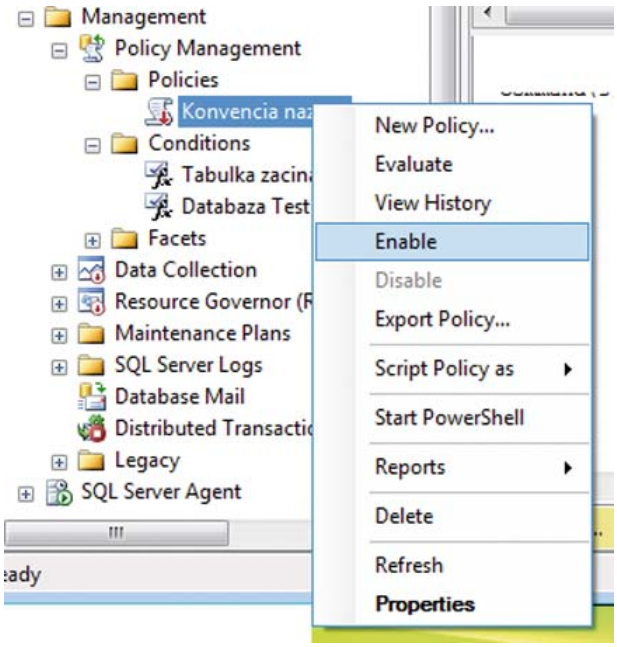
Ak je reštrikčná politika zapnutá, v takom prípade sa nám to nepodarí, server vyhlási chybu – porušenie politiky

```
Policy 'Table Naming Conventions' has been violated by '/Server/(local)/Database/
pokusy/Table/dbo.zamestnanci'.
This transaction will be rolled back.
Policy description: ''
Additional help: '': ''.
Msg 3609, Level 16, State 1, Procedure sp_syspolicy_dispatch_event, Line 50
The transaction ended in the trigger. The batch has been aborted.
```

Príkaz, ktorý je v súlade s politikou prebehne úspešne:

```
CREATE TABLE tblZamestnanci
(
   ID int NOT NULL,
   meno nvarchar(30),
   email nvarchar(30))
```

Vopred nastavené politiky môžeme podľa potreby zapnúť a vypnúť v kontextovom menu.



Jednotlivé nastavené politiky môžeme podľa potreby zapnúť a vypnúť

Kapitola 7: Nové údajové typy pre dátum a čas

Vo verzii 2005 disponoval SQL Server len jedným údajovým typom DATETIME pre uloženie hodnôt dátumu a času. Verzia 2008 prináša nové údajové typy:

Date pre uloženie dátumu v rozmedzí 1. 1. 0001 do 31. 12. 9999 Gregoriánskeho kalendára,

TimeE pre uloženie času s presnosťou na 100 nanosekúnd,

DateTimeOffset – UTC dátum a čas. UTC (Koordinovaný svetový čas, po anglicky Universal Time Coordinated) je čas definovaný na základe medzinárodného atómového času upravovaný v pravidelných intervaloch skokom o 1 sekundu tak, aby sa od astronomického času UT1 neodlišoval o viac ako 0,7 sekundy. Tento čas celosvetovo udržuje a koordinuje časová sekcia BIPM Paríž. SR je od času UTC posunutá o + 1 hod,

DateTime2 – veľký dátumový rozsah s presnosťou 100 nanosekúnd, v rozmedzí 1. 1. 0001 00:00:00,0000000 do 31. 12. 9999 23:59:59,9999999 Gregoriánskeho kalendára,

Smaldatetime – dátum a čas v rozsahu 1. 1. 1900 do 6. 6. 2079 s presnosťou 1 minúty,

Datetime – pôvodný údajový typ používaný vo verziách SQL Server 2005 a 2000 umožňuje dátum a čas v rozsahu 1. 1. 1753 do 31. 12. 9999 presnosťou 0,00333

Data type	Format	Range	Accuracy	Storage size (bytes)	User-defined fractional second precision	Time zone offset
time	hh:mm:ss [.nnnnnnn]	00:00:00.0000000 through 23:59:59.9999999	100 nanoseconds	3 to 5	Yes	No
date	YYYY-MM-DD	00001-01-01 through 9999-12-31	1 day	3	No	No
smalldatetime	YYYY-MM-DD hh:mm:ss	1900-01-01 through 2079-06-06	1 minute	4	No	No
datetime	YYYY-MM-DD hh:mm:ss [.nnn]	1753-01-01 through 9999-12-31	0.333 second	8	No	No
datetime2	YYYY-MM-DD hh:mm:ss [.nnnnnnn]	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999	100 nanoseconds	6 to 8	Yes	No
datetimeoffset	YYYY-MM-DD hh:mm:ss [.nnnnnnn] [+|-] hh:mm	00001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 (in UTC)	100 nanoseconds	8 to 10	Yes	Yes

Prehľadná tabuľka údajových typov pre dátum a čas

Ak chceme zistiť dostupné údajové typy, môžeme položiť dopyt do systémového pohľadu SYS.SYSTYPES. Nakoľko nás zaujímajú len tie údajové typy, ktoré majú v názve slovíčko DATE alebo TIME, sformulujeme podmienku v klauzule WHERE:

```
SELECT name, length FROM sys.systypes
  WHERE name LIKE '%date%' OR name LIKE '%time%'

name                             length
-------------------------------- ------
date                             3
time                             5
datetime2                        8
datetimeoffset                   10
smalldatetime                    4
datetime                         8
timestamp                        8

(7 row(s) affected)
```

Pre demonštráciu použitia týchto údajových typov vytvoríme novú cvičnú tabuľku, kde pre nové typy pre ukladanie hodnôt dátumu a času vytvoríme jeden atribút:

```
CREATE TABLE datove_typy
(
  d1 DATE,
  d2 TIME(3),
  d3 DATETIME2(7) NOT NULL DEFAULT GETDATE(),
  d4 DATETIMEOFFSET CHECK (d4 < CAST(GETDATE() AS DATETIMEOFFSET(0)))
);
```

Aby sme ukázali, ako sa s jednotlivými údajovými typmi pracuje uložíme do cvičnej tabuľky nejaké údaje, pre zaujímavosť uložíme limitné hodnoty, teda minimálny a maximálny dátumový a časový údaj:

```
INSERT INTO datove_typy
VALUES ('0001-01-01', '23:59:59',
    '0001-12-21 23:59:59.1234567',
    '0001-10-21 23:59:59.1234567 -07:00');

INSERT INTO datove_typy
VALUES ('9999-12-31', '23:59:59',
    '9999-12-31 23:59:59.1234567',
    '1111-10-21 23:59:59.1234567 -07:00');
```

V poslednom kroku sa pokúsime tieto údaje vypísať:

```
SELECT d4, DATEPART(TZOFFSET, d4),
    DATEPART(ISO_WEEK, d4),
    DATEPART(MICROSECOND, d4)
FROM datove_typy;
```

```
d4
---------------------------------- ----------- ----------- -----------
0001-10-21 23:59:59.1234567 -07:00 -420        42          123456
1111-10-21 23:59:59.1234567 -07:00 -420        42          123456

VISTAVYVOJ-PC(VistaVyvoj-PC\VistaVyvoj): (2 row(s) affected)
```

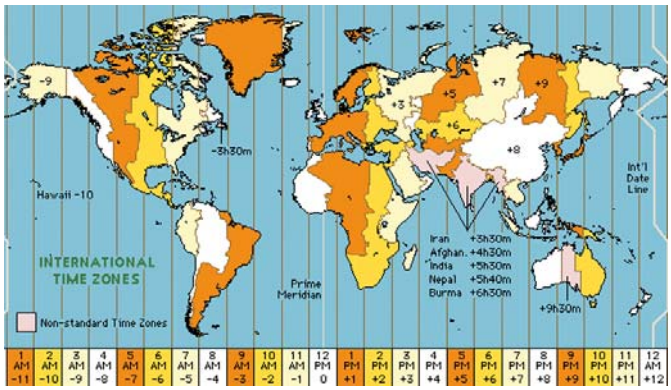
Pre rekapituláciu ukážeme prevod konkrétnej hodnoty dátumu a času na prezentované údajové typy:

```
SELECT
     CAST('2007-05-08 12:35:29. 1234567 +12:15' AS time(7)) AS 'time'
    ,CAST('2007-05-08 12:35:29. 1234567 +12:15' AS date) AS 'date'
    ,CAST('2007-05-08 12:35:29.123' AS smalldatetime) AS
        'smalldatetime'
    ,CAST('2007-05-08 12:35:29.123' AS datetime) AS 'datetime'
    ,CAST('2007-05-08 12:35:29. 1234567 +12:15' AS datetime2(7)) AS
        'datetime2'
    ,CAST('2007-05-08 12:35:29.1234567 +12:15' AS datetimeoffset(7)) AS
        'datetimeoffset';
```

Výsledky konverzie budú v tvare:

```
time                12:35:29. 1234567
date                2007-05-08
smalldatetime       2007-05-08 12:35:00
datetime            2007-05-08 12:35:29.123
datetime2           2007-05-08 12:35:29. 1234567
datetimeoffset      2007-05-08 12:35:29.1234567 +12:15
```

Nakoniec ukážeme výpis času pre jednotlivé časové zóny. Vzhľadom na globálnu ekonomiku a filiálky firiem po celom svete je čas podľa časových pásiem veľmi dôležitý. Preto nezaškodí malé zopakovanie zemepisu ohľadne časových pásiem a medzinárodnej dátumovej čiary.
Naša zemeguľa je rozdelená na 24 časových pásiem, ktoré sú vzájomne posunuté o jednu hodinu. Toto rozdelenie má počiatok na takzvanom nultom poludníku, ktorý prechádza cez Greenwich. Čas, ktorý je platný na tomto nultom poludníku je medzinárodne akceptovaný čas GMT (Greenwich Mean Time), alebo inak nazývaný UTC. Delenie na časové pásma nie je presné a berú sa do úvahy hranice štátov a podobne. Veľká časť kontinuálnej Európy vrátane Čiech a Slovenska leží v pásme GMT +1. Pri ceste na západ sa čas posúva o hodinu až v Portugalsku. Celý princíp je možné najľahšie pochopiť z názorného obrázka.



Mapa časových pasiem

Okrem nultého poludníka ako počiatku rozdelenia zemegule na časové zóny je významná aj medzinárodná dátumová čiara (na našom obrázku vpravo), ktorá prebieha prevažne neobývanými oblasťami Tichého oceánu a približne sleduje 180. poludník. Pri prekročení tejto pomyslenej hranice sa posúva dátum. V praxi to vyzerá tak, že ako prví majú nový dátum obyvatelia Fidži, Marshallových ostrovov, potom nasleduje Nový Zéland, Austrália, Sibír. O päť hodín začína nový deň v Indii, o 11 hodín v Strednej Európe a až o 16 hodín začne nový deň na východnom pobreží Spojených štátov.

Aj pre tento príklad vytvoríme cvičnú tabuľku, v ktorej budeme čas ukladať do atribútu s údajovým typom. DATETIMEOFFSET:

```
create table tbl_timezones
(
   id int identity(1,1),
   cas datetimeoffset)

insert into tbl_timezones (cas) values ('1998-09-20 7:45:50.71345 -5:00')
insert into tbl_timezones (cas) values ('1956-01-27 6:45:50.00000 -3:00')
insert into tbl_timezones (cas) values ('1972-12-18 7:45:50.71345 +1:00')
insert into tbl_timezones (cas) values ('2005-01-20 7:12:50.71345 +9:00')
insert into tbl_timezones (cas) values ('2005-01-20 01:00:00.00000 +4:00')
```

pre výpis časového posunutia využijeme funkciu DATEPART, ktorej úlohou je vypísať časť dátumu a času. Jej syntax je:

DATEPART(časť_dátumu, dátum).

Funkcia vráti číselnú hodnotu časti dátumu, zadanú prvým parametrom, pre dátum zadaný druhým parametrom. Napríklad, zistíme aktuálny mesiac:

SELECT DATEPART(month, GETDATE()) FROM test; [2]

V našom prípade vytvoríme dopyt pre výpočet offsetu. Najskôr zistíme lokálny ofset (v minútach) pre aktuálne časové pásmo voči GMT:

```
select datepart(TZoffset,sysdatetimeoffset())


60
```

Naše posunutie voči GMT je 60 minút teda GMT + 1 hodina. Vypíšme nami zadané časy tak, aby sme vedeli aký bol v tom okamihu čas v našom časovom pásme. V ľavom stĺpci je čas v jeho vlastnej časovej zóne, v pravom stĺpci čas v našom časovom pásme GMT +1

```
SELECT cas, SWITCHOFFSET (cas, datepart(TZoffset,sysdatetimeoffset()))
AS TimeinCurrentTimezone
FROM tbl_timezones

as                                TimeinCurrentTimezone
--------------------------------- ----------------------------------
20. 9. 1998 7:45:50 -05:00         20. 9. 1998 13:45:50 +01:00
27. 1. 1956 6:45:50 -03:00         27. 1. 1956 10:45:50 +01:00
18. 12. 1972 7:45:50 +01:00        18. 12. 1972 7:45:50 +01:00
20. 1. 2005 7:12:50 +09:00         19. 1. 2005 23:12:50 +01:00
20. 1. 2005 1:00:00 +04:00         19. 1. 2005 22:00:00 +01:00

(5 row(s) affected)
```

Kapitola 8: Údajový typ TABLE

Table-valued je nový používateľom definovaný údajový typ umožňujúci ukladať tabuľkové údaje. Najlepšie si jeho použitie ukážeme na praktickom príklade. Najskôr ukážeme štandardný spôsob vkladania nových záznamov cez parametrickú uloženú procedúru. Vytvoríme cvičnú tabuľku zamestnancov, do ktorej budeme pridávať údaje:

```
CREATE TABLE zamestnanci
(
   ID int NOT NULL,
   meno nvarchar(30),
   email nvarchar(30))
```

Následne vytvoríme „klasickú" parametrickú uloženú procedúru pre pridávanie záznamov:

```
CREATE PROCEDURE MultiZadavanie(@id int, @meno nvarchar(30),
@email nvarchar(30))
AS
BEGIN
INSERT INTO dbo.zamestnanci
values(@id, @meno, @email)
END
```

Pomocou uloženej procedúry pridáme niekoľko záznamov:

```
execute MultiZadavanie 1, 'Mickey Mouse','mickey@disney.com'
execute MultiZadavanie 2, 'Donald Duck','donald@disney.com'
execute MultiZadavanie 3, 'Snehurka','snehurka@disney.com'
```

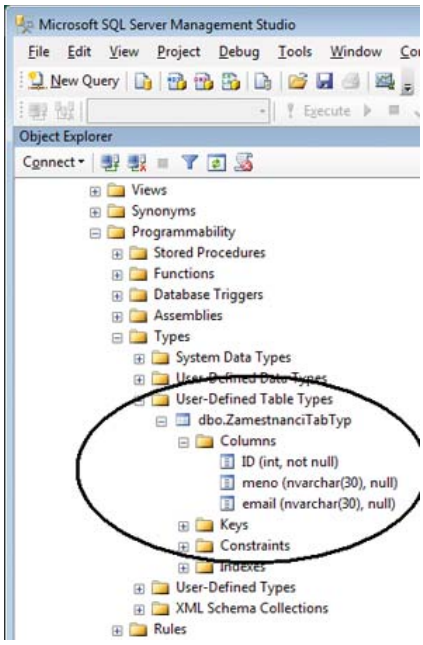
Tabuľka bude obsahovať tieto záznamy:

```
ID          meno                          email
----------- ------------------------------ ------------------------------
1           Mickey Mouse                  mickey@disney.com
2           Donald Duck                   donald@disney.com
3           Snehurka                      snehurka@disney.com
```

Rovnakú funkcionalitu teraz vykonáme pomocou nového údajového typu TABLE. Vytvoríme používateľsky definovaný údajový typ TABLE s vhodnou štruktúrou pre ukladanie údajov o zamestnancoch:

```
CREATE TYPE ZamestnanciTabTyp AS TABLE
( ID int NOT NULL, meno nvarchar(30), email nvarchar(30))
```

O vytvorení údajového typu TABLE sa môžeme presvedčiť pomocou nástroja SQL Server Management Studia. Nájdeme ho v záložke Programability –> Types –> User-Defined Table Types.



Používateľom definovaný údajový typ TABLE v nástroji Management Studio

Vytvoríme uloženú procedúru pre pridávanie záznamov s využitím údajového typu TABLE

```
CREATE PROCEDURE MultiZadavanie2(@zamestnanec ZamestnanciTabTyp READONLY)
AS
BEGIN
INSERT INTO dbo.zamestnanci
SELECT * FROM @zamestnanec
END
```

Vytvoríme dočasnú premennú nami definovaného tabuľkového údajového typu TABLE, naplníme ju údajmi a pomocou uloženej procedúry uložíme jej obsah do databázovej tabuľky zamestnanci:

```
DECLARE @zamestnanci ZamestnanciTabTyp

INSERT INTO @zamestnanci
VALUES (1, 'Mickey Mouse','mickey@disney.com')

INSERT INTO @zamestnanci
VALUES (2, 'Donald Duck','donald@disney.com')

INSERT INTO @zamestnanci
VALUES (3, 'Snehurka','snehurka@disney.com')

EXECUTE MultiZadavanie2 @zamestnanci
```

Ak porovnáme obidve metódy, vidíme, že v druhom prípade sa uložená procedúra volala namiesto trikrát len raz. Taktiež v prípade zmeny štruktúry tabuľkových údajov, nie je nutné rekompilovať uloženú procedúru. Tento spôsob je výhodnejší, jednoduchší a rýchlejší aj v prípade ak programujeme vyššiu vrstvu v .NET jazyku v aplikácii Visual Studio. Tento údajový typ je plne podporovaný aj v ADO.NET 3.0.

```
static void Main(strings[] args)
{
    string cnStr = settings.ConnectionString;
    using (SqlConnection connection = new SqlConnection())
    {
       connection.ConnectionString =cnStr;
       DataTable dt = GetDataTable();
       SqlCommand cmd = new SqlCommand();
       cmd.Connection = connection;
       cmd.CommandText = „MultiZadavanie2";
       cmd.CommandType = System.Data.CommandType.StoredProcedure;
       SqlParameter tvpParam = cmd.Parameters.AddWithValue(„@ZamDetaily", dt);
       tvpParam.SqlDbType = SqlDbType.Structured;
       tvpParam.TypeName = „ZamestnanciTabTyp"
       connection.Open();
       cmd.ExecuteNonQuery();
    }


private static DataTable GetDataTable()
{
   DataTable dt = nev DataTable(„emp");

   dt.Columns.Add(„ID", System.Type.GetType(„System.Int32"));
   dt.Columns.Add(„meno", System.Type.GetType(„System.String"));
   dt.Columns.Add(„email", System.Type.GetType(„System.String"));

   dt.Rows.Add(CreateRow(dt, 1, 'Mickey Mouse','mickey@disney.com'));
   dt.Rows.Add(CreateRow(dt, 2, 'Donald Duck','donald@disney.com'));
   dt.Rows.Add(CreateRow(dt, 3, 'Snehurka','snehurka@disney.com'));
   return dt
}
```

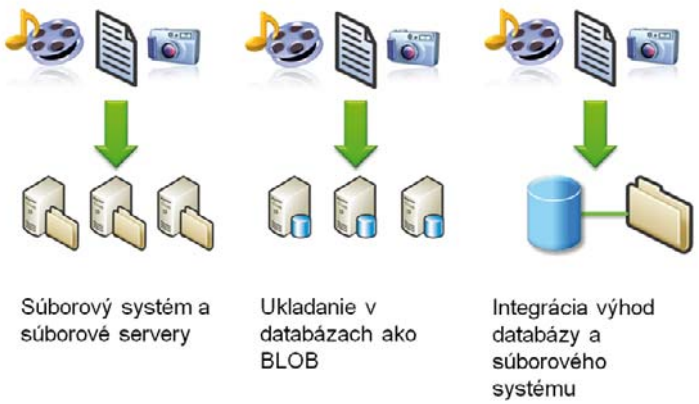
Nový používateľom definovaný údajový typ TABLE je výhodný najmä pre dávkové aktualizácie, migráciu údajov a pri veľkom objeme údajov posielaných z klienta na server.

Kapitola 9: Údajový typ FILESTREAM

Z grafu názorne vidieť, že viac než 93 % informácií bolo získaných a uložených v digitálnej podobe v posledných rokoch. Prevažná väčšina z nich je uložených v netextovej, teda prevažne multimediálnej podobe ako digitálne fotografie, zvukové a obrazové záznamy. Len 0,003 % z informácií je uložených v textovom (nie hypertextovom) formáte. Ďalšou nemenej zaujímavou skutočnosťou je, že približne 30 % zo zhromaždených údajov sú momentálne uložené na diskoch bežných PC so všetkými výhodami, ale hlavne nevýhodami, ktoré sú s tým spojené. Všetci si živo viete predstaviť tú obrovskú duplicitu údajov, minimálnu bezpečnosť, a na druhej strane zložité vyhľadávanie a komplikovaný prístup k informáciám. Ďalších 30 % údajov je uložených na digitálnych nosičoch (CD, DVD...) a len približne 15 až 30 % údajov je bezpečne uložených na serveroch v databázach. Čoraz populárnejší sa stáva formát XML a to nielen pre výmenu údajov, ale aj pre ich uloženie. Jeho masívny nástup začal s rozvojom webových služieb.

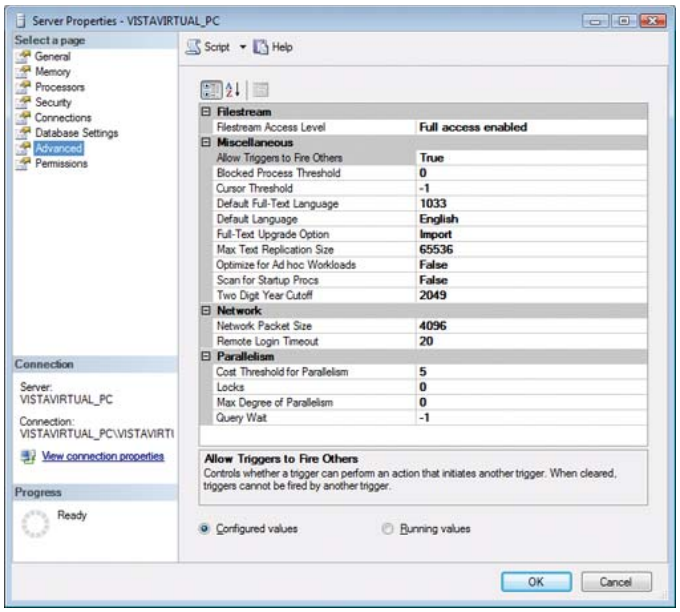
Objemy údajov skladovaných a spravovaných v databázach a súboroch rastú geometrickým radom

Nový údajový typ Filestream podporuje uloženie veľkých binárnych objektov do NTFS filesystému, zatiaľ čo tieto objekty zostávajú nedeliteľnou súčasťou databázy.



Ukladanie údajov v databázach, súboroch a integrácia databázy a súborového systému

Pred prvým použitím je potrebné povoliť používanie údajového typu FILESTREAM na úrovni databázového engine. Môžeme to urobiť v záložke „Advanced" databázového servera.



Povolenie používania údajového typu FILESTREAM

alebo pomocou SQL príkazu spúšťajúceho uloženú procedúru `sp_filestream_configure`.

```
USE master;
GO

EXEC sp_filestream_configure @enable_level = 3;
GO
```

Parameter enable_level môže nadobúdať hodnoty:

0 – implicitná hodnota, kedy je FILESTREAM zakázaný,
1 – Prístup k údajom v údajovom type FILESTREAM je len pre Transact-SQL,
2 – Prístup k údajom v údajovom type je pre Transact-SQL a lokálny súborový systém,
3 – Prístup k údajom v údajovom type je pre Transact-SQL, lokálny a vzdialený súborový systém.

Musíme byť prihlásení s právami administrátora, inak budeme na túto skutočnosť upozornení pomocou chybového hlásenia.

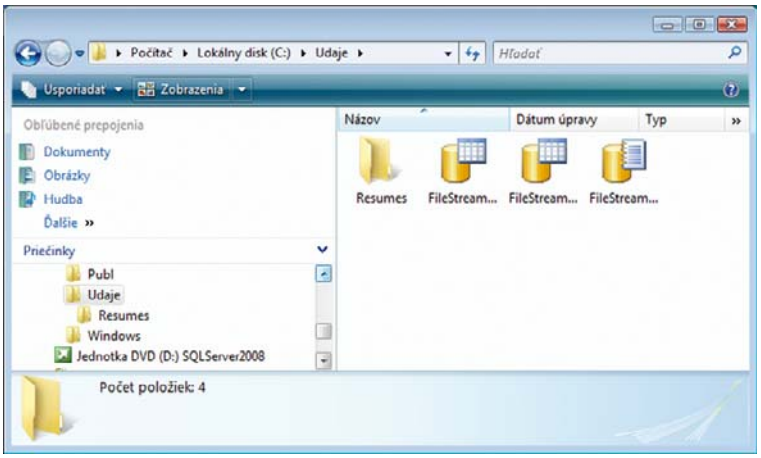
```
VISTAVYVOJ-PC: Msg 5587, Level 14, State 1, Procedure sp_filestream_configureYou do not
have permissions to configure FILESTREAM feature.
You need Windows Administrator and sysadmin rights to configure FILESTREAM feature.
```

Následne vytvoríme databázu, ktorá bude využívať FILESTREAM. Súbory budú na disku C v priečinku UDAJE

```
CREATE DATABASE FileStreamDB ON PRIMARY
  ( NAME = FileStreamDB_data,
    FILENAME = N'C:\udaje\FileStreamDB_data.mdf',
    SIZE = 10MB,
    MAXSIZE = 50MB,
    FILEGROWTH = 15%),
FILEGROUP RowGroup1
  ( NAME = FileStreamDB_group1,
    FILENAME = N'C:\udaje\FileStreamDB_group1.ndf',
    SIZE = 10MB,
    MAXSIZE = 50MB,
    FILEGROWTH = 5MB),
```

```
FILEGROUP FileStreamGroup1 CONTAINS FILESTREAM
  ( NAME = FileStreamDBResumes,
    FILENAME = N'C:\udaje\Resumes')
LOG ON
  ( NAME = 'FileStreamDB_log',
    FILENAME = N'C:\udaje\FileStreamDB_log.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB);
GO
```

Po úspešnom vytvorení databázy sa môžeme pozrieť do priečinka UDAJE, či v ňom vznikli príslušné súbory a podpriečinok RESUMES



Súbory v priečinku databázy využívajúcej údajový typ FILESTREAM

Parametre novovytvorenej databázy si môžeme pozrieť aj v nástroji SQL Server Management Studio

V nami vytvorenej databáze ktorá bude využívať FILESTREAM, vytvoríme jednoduchú databázovú tabuľku:

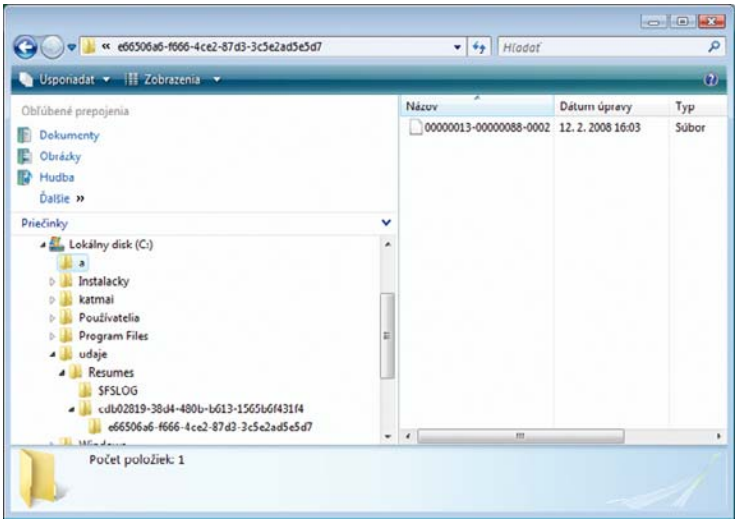
```
USE FileStreamDB;
GO

CREATE TABLE dbo.katalog
(katalog_id UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL UNIQUE,
  nazov varchar(25),
  Resume varbinary(max) FILESTREAM);
GO
```

Ak vložíme záznam s hodnotu atributú 'Resume' NULL, nebude pre tento záznam vytvorený žiadny súbor:

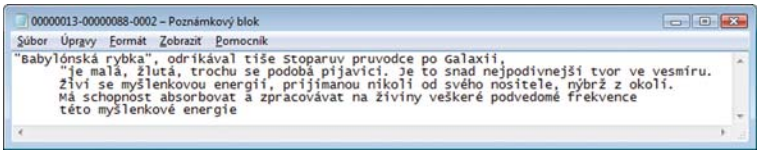
```
INSERT INTO dbo.katalog
 VALUES (newid (), 'prazdny', NULL);
```

Teraz vložme záznam obsahujúci text, napríklad:

```
INSERT INTO dbo.katalog
    VALUES (newid (), 'Babylonska rybka',
      CAST ('"Babylónská rybka", odříkával tiše Stopařův průvodce po Galaxii,
      „je malá, žlutá, trochu se podobá pijavici. Je to snad nejpodivnější
      tvor ve vesmíru. Živí se myšlenkovou energií, přijímanou nikoli
      od svého nositele, nýbrž z okolí.
      Má schopnost absorbovat a zpracovávat na živiny veškeré podvědomé frekvence
      této myšlenkové energie' as varbinary(max)));
```



Údaje z údajového typu FILESTREAM sú uložené do súboru

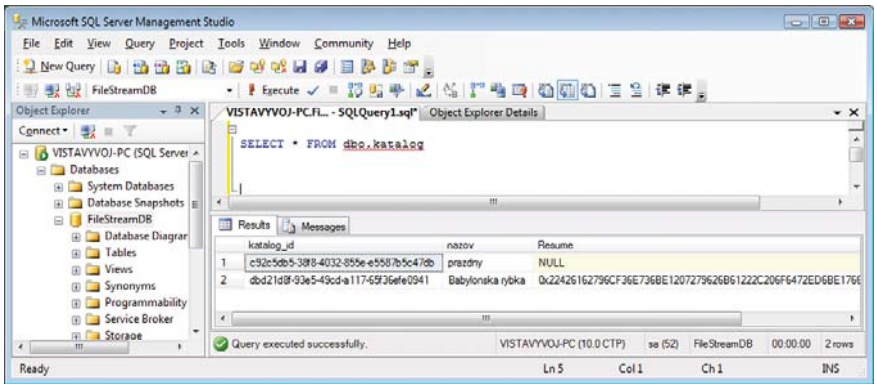


Obsah súboru

Pre zaujímavosť môžeme skúsiť dopyt:

```
SELECT * FROM dbo.katalog
```

Dopyt vráti názov a hexadecimálnu prezentáciu obsahu.



Výpis databázovej tabuľky

Aplikačné rozhranie pre FILESTREAM:

```
// New TSQL Function: Get_filestream_transaction_context()
       SqlTransaction Tran = sqlConn.BeginTransaction();
       SqlCommand cmd = new SqlCommand(SELECT  Get_filestream_transaction_
context(),conn);
       cmd.Transaction = Tran;
       byte[] txCxt = (byte[]) cmd.ExecuteScalar();

 // New TSQL Function: PathName()
       SqlCommand cmd = new SqlCommand(SELECT Resume.PathName()    FROM Demo_DB..
student", conn);
       string path = (string) cmd.ExecuteScalar();

// New SQL Native Client Function: OpenSqlFilestream()
HANDLE hImage = OpenSqlFilestream(path,..., txCxt);
       WriteFile(hImage,...);

// All triggers will be fired upon closing the handle
CloseHandle(hImage);
```

Kapitola 10: Príkaz MERGE pre synchronizáciu obsahu tabuliek

Novinkou vo verzii 2008 je aj príkaz MERGE. Je v zhode so štandardom SQL 2006. Slúži na synchronizáciu obsahu tabuliek. Ak pri vkladaní záznamu zo zdrojovej do cieľovej tabuľky takýto záznam ešte neexistuje, uloží sa do databázy, alebo do údajového skladu ako nový. Ak záznam už existuje, tak sa pomocou príkazu MERGE aktualizuje už existujúci záznam. Pomocou tohto príkazu môžeme nahradiť sekvenciu viacerých príkazov typu INSERT, UPDATE, DELETE, nehovoriac už o zisťovaní či taký záznam existuje, prípadne o ošetrovaní výnimky pri klasickom vkladaní už existujúceho záznamu.

Príkaz MERGE spája dve tabuľky podľa kritéria:

MERGE target USING source ON

Podľa výsledku spojenia vykonáva INSERT/UPDATE/DELETE

WHEN MATCHED THEN
  Riadky spĺňajúce kritéria. (INNER JOIN)
WHEN [TARGET] NOT MATCHED THEN
  Nevyhovujúce riadky v TARGET tabuľke (LEFT OUTER JOIN)
WHEN NOT MATCHED BY SOURCE THEN
  Nevyhovujúce riadky v SOURCE tabuľke (RIGHT OUTER JOIN)

V príkaze môžeme využívať dodatočné klauzuly
 WHEN MATCHED AND tab.aValue > 100 THEN
 WHEN MATCHED AND tab.aValue < 100 THEN

Použitie príkazu MERGE ukážeme na praktickom príklade. Pomocou príkazu CERATE TABLE vytvoríme dve tabuľky – zdrojovú a cieľovú, ktoré budeme synchronizovať.

```
CREATE TABLE zdroj (id INT, meno NVARCHAR(20), skore_hry INT);
CREATE TABLE ciel (id INT, meno NVARCHAR(20), skore_hry INT);
```

Do nich vložíme niekoľko záznamov

```
INSERT INTO zdroj VALUES (1, 'Peter', 100);
INSERT INTO zdroj VALUES (2, 'Ignac', 200);
INSERT INTO ciel VALUES (1, 'Mirka', 100);
INSERT INTO ciel VALUES (2, 'Ignac', 100);
```

Tabuľky obsahujú údaje

zdroj

```
id          meno                 skore_hry
----------- -------------------- -----------
1           Peter                100
2           Ignac                200
```

cieľ

```
id          meno                 skore_hry
----------- -------------------- -----------
1           Mirka                100
2           Ignac                100
```

Pomocou príkazu MERGE zosynchronizujeme cieľovú tabuľku so zdrojovou:

```
MERGE ciel AS c
    USING zdroj as z
    ON c.id = z.id
    WHEN MATCHED AND
       (c.meno != z.meno OR c.skore_hry!= z.skore_hry) THEN
        -- Ak riadok existuje no obsahuje rozdielne udaje
        UPDATE SET c.meno = z.meno, c.skore_hry = z.skore_hry
    WHEN NOT MATCHED THEN
        -- Ak riadok existuje len v zdrojovej tabulke
       INSERT VALUES (z.id, z.meno, z.skore_hry)
    WHEN NOT MATCHED BY SOURCE THEN
        -- Ak riadok existuje len v cielovej tabulke vymažeme ho
        DELETE
    OUTPUT $action, inserted.id, deleted.id;
```

V poslednom riadku príkazu sme nechali vypísať ID vložených a vymazaných záznamov:

```
$action    id          id
---------- ----------- -----------
UPDATE     1           1
UPDATE     2           2
```

Po vykonaní príkazu MERGE naše cvičné tabuľky obsahujú údaje:

zdroj

```
id          meno                 skore_hry
----------- -------------------- -----------
1           Peter                100
2           Ignac                200
```

cieľ

```
id          meno                 skore_hry
----------- -------------------- -----------
1           Peter                100
2           Ignac                200
```

Kapitola 11: XML

Technológia XML je otvorená a platformovo nezávislá. Definuje formát údajov a operácie s nimi. Zjednodušene by sa dalo povedať, že tento formát pre výmenu údajov je rovnako dobre čitateľný pre počítače aj pre ľudí. Tieto údaje môžeme ľahko načítať a identifikovať pomocou pomerne jednoduchej rutiny v ľubovoľnom programovacom jazyku.

Formát XML sa stal kľúčovou technológiou pre výmenu informácií medzi „business" aplikáciami, aplikáciami typu ERP a CRM, používa sa u webových služieb a podobne. V aplikáciách tohto typu je technológia XML doslova priemyselným štandardom. Možnosti a rozsah využívania XML technológie vzrástli natoľko, že ide dalo by sa povedať o „typovú revolúciu". Uložiť údaje v XML formáte do databázy na ľubovoľnej databázovej platforme nie je príliš ťažké (znalcom XML nemusíme pripomínať, že sa jedná v podstate o štruktúrovaný textový formát). Technologická vyspelosť a miera úspešnosti implementácie XML u tej – ktorej platformy sa naplno prejaví až pri požiadavke na čítanie a vyhľadávanie veľkého množstva údajov, požiadavke na bezpečnosť a konzistenciu údajov, požiadavke na 100percentnú kompatibilitu podľa odporúčaní konzorcia W3C, na efektívnu podporu práce so štruktúrovanými aj neštruktúrovanými údajmi súčasne a podobne. V SQL Serveri je podpora XML integrovaná priamo do jazyka SQL,

Už SQL Server vo verzii 2000 obsahoval klauzulu FOR XML pre výpis obsahu databázovej tabuľky vo formáte XML a klauzulu OPENXML pre načítanie časti XML dokumentu. Klauzula FOR XML sa využívala v príkaze SELECT:

```
SELECT * FROM tabulka FOR XML mode [, XMLDATA] [, ELEMENTS][, BINARY BASE64]
```

pričom:
- **mode**.. určuje tvar výsledku, prípustné hodnoty sú RAW, AUTO, EXPLICIT,
- **XMLDATA**.. určuje, že súčasťou dokumentu bude aj XML schéma,
- **ELEMENTS**.. v móde AUTO sa určuje, že údaje budú dávané ako elementy. Inak sú údaje v hodnotách atribútov,
- **BINARY BASE64**.. určuje, že binárne dáta budú reprezentované vo formáte base64-encoded.

Skôr než začneme preberať možnosti práce s XML, vytvoríme si a naplníme jednoduchú cvičnú tabuľku „Zákazníci".

```
CREATE TABLE zakaznici
(
       id_zak int PRIMARY KEY,
       nazov varchar(20) NOT NULL,
       obrat money,
);
```

```
Policy 'Table Naming Conventions' has been violated by '/Server/(local)/Database/
pokusy/Table/dbo.Zakaznici'.
This transaction will be rolled back.
Policy description: ''
Additional help: '': ''.
Msg 3609, Level 16, State 1, Procedure sp_syspolicy_dispatch_event, Line 50
The transaction ended in the trigger. The batch has been aborted.
```

Ale, ale, vari sme zabudli, že sme si cez Declarative Managemet Framework nastavili politiku pre názvy tabuliek v databáze „pokusy". Názov tabuľky predsa musí začínať prefixom „tbl". Takže znovu a tentoraz už v súlade s nastavenou politikou:

```
CREATE TABLE tblZakaznici
(
       id_zak int PRIMARY KEY,
       nazov varchar(20) NOT NULL,
       obrat money,
);
```

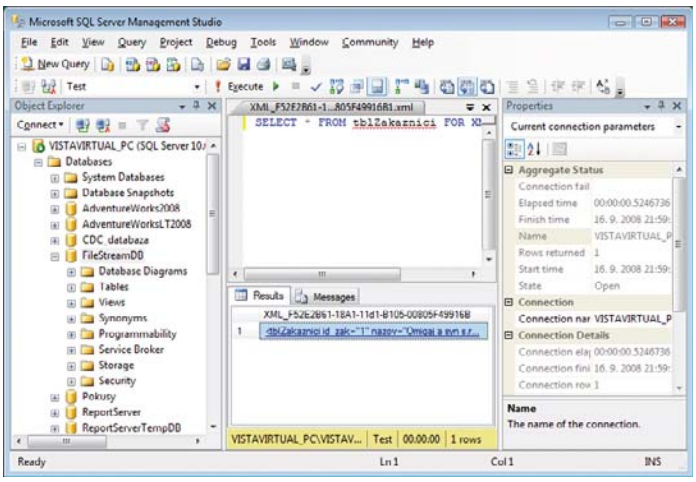
```
INSERT INTO tblZakaznici
VALUES(1, 'Omigaj a syn s.r.o', 365000);

INSERT INTO tblZakaznici
VALUES(2, 'Postavil & Rozboril', 2000000);

INSERT INTO tblZakaznici
VALUES(3, 'Prva distribucna a.s', 1565000);
```

Teraz môžeme vypísať údaje vo formáte XML:

```
SELECT * FROM tblZakaznici FOR XML AUTO
```

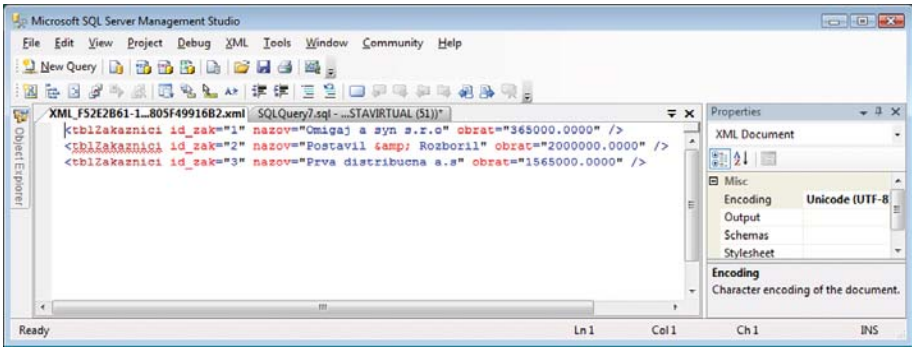
```
<zakaznici id_zak="1" nazov="Omigaj a syn s.r.o" obrat="365000.0000"/>
<zakaznici id_zak="2" nazov="Postavil & Rozboril" obrat="2000000.0000"/>
<zakaznici id_zak="3" nazov="Prva distribucna a.s" obrat="1565000.0000"/>
```

Takto sa vypíše obsah v textovom móde. Ak prepneme konzolu do tabuľkového výstupu, zobrazí sa prepojenie na XML dokument.



Výpis XML dokumentu ako prepojenia

Kliknutím na prepojenie otvoríme dokument v novej záložke.



Obsah XML dokumentu v novej záložke

Ukážeme aj príklad využitia OPENXML:

```
DECLARE @doc nvarchar(1000)
SET @doc = '<Objednavka OID = „1011“>
                    <Polozka PID=“Chlieb“ Pocet=“20“/>
                    <Polozka PID=“Rohliky“ Pocet=“100“/>
             </Objednavka>'
DECLARE @xmlDoc integer
EXEC sp_xml_preparedocument @xmlDoc OUTPUT, @doc
SELECT * FROM
OPENXML (@xmlDoc, 'Objednavka/Polozka', 1)
WITH
(OID integer '../@OID',
 PID varchar(10),
 Pocet integer)
EXEC sp_xml_removedocument @xmlDoc
```

```
OID         PID        Pocet
----------- ---------- -----------
1011        Chlieb     20
1011        Rohliky    100
```

Uložené procedúry sp_xml_preparedocument a sp_xml_removedocument slúžia na vytvorenie a zrušenie XML dokumentu v dočasnej premennej v pamäti.

Pri použití klasifikátora PATH vo verzii SQL Server 2005 mená stĺpcov určujú XPATH polohu v XML stromovej štruktúre:

```
SELECT id_zak AS '@ID', nazov AS 'Nazov'
FROM tblZakaznici
FOR XML PATH ('Zakaznici'), ROOT ('Zakaznici')
```

```
<Zakaznici>
  <Zakaznici ID=“1“>
    <Nazov>Omigaj a syn s.r.o</Nazov>
  </Zakaznici>
  <Zakaznici ID=“2“>
    <Nazov>Postavil &amp; Rozboril</Nazov>
  </Zakaznici>
  <Zakaznici ID=“3“>
    <Nazov>Prva distribucna a.s</Nazov>
  </Zakaznici>
</Zakaznici>
```

Natívny XML údajový typ

Po oboznámení sa s možnosťou výstupu údajov z bežných tabuliek vo formáte XML podrobnejšie predstavíme natívny XML údajový typ. Môžeme ho použiť ako:
- typ stĺpca v tabuľke,
- typ parametra v uloženej procedúre,
- typ návratovej hodnoty v „user-defined function“,
- typ premennej.

Nakoľko aj keď je XML dokument vo formáte textu, XML dátový typ nie je takéhoto typu a preto niektoré operácie bežné u textových dátových typov nepodporuje. Okrem toho, že stĺpec XML dátového typu nie je možné použiť ako primárny kľúč, nepodporuje textové porovnávanie, operátory GROUB BY a ORDER BY a obmedzenie UNIQUE.

Pre prvé pokusy s údajovým typom XML si vytvoríme pre tento údajový typ premennú:

```
DECLARE @premenna xml
SET @premenna='<A></A><B>bbb</B>'
SELECT @premenna AS premenna
SELECT CONVERT(nvarchar(1000),@premenna) AS premenna_ako_retazec
```

Všimnite si, že reprezentácia prázdneho tagu je odlišná od vstupného reťazca:

```
premenna
------------------------
<A /><B>bbb</B>

premenna_ako_retazec
------------------------
<A/><B>bbb</B>
```

Môžeme sa pokúsiť databázovému serveru podvrhnúť aj non-well formatted XML dokument, kde počiatočný a koncový element nie sú rovnaké:

```
DECLARE @premenna xml
SET @premenna='<A></AB>'
```

no skončí to samozrejme nezdarom:

```
Msg 9436, Level 16, State 1, Line 2
XML parsing: line 1, character 8, end tag does not match start tag
```

Pre demonštráciu základných možností práce s XML dokumentmi a natívnym XML údajovým typom vytvoríme jednoduchú cvičnú tabuľku, ktorá obsahuje dva stĺpce. V jednom je ID záznamu ako údajový typ integer, druhý stĺpec bude obsahovať natívny XML údajový typ:

```
CREATE TABLE tblXml_tabulka
(
   id INT,
   xml_stlpec XML)
```

Do takto navrhnutej tabuľky vložme jeden záznam. Všimnite si, že do údajového typu XML vkladáme textový reťazec. Je to v poriadku, však XML dokument je dokument textovej povahy. V priebehu vkladania prebehne validácia a auto konverzia textového reťazca na XML dokument:

```
INSERT INTO tblXml_tabulka VALUES(1, '<doc/>')
```

Ak textový reťazec neobsahuje platný XML dokument (tagy X1 a X2 sú vo vnútri dokumentu navzájom prehodené), autokonverzia neprebehne a príkaz na vkladanie záznamov skončí chybovým hlásením:

```
INSERT INTO tblXml_tabulka VALUES(2, '<doc><x1><x2></x1></x2></doc>')

Msg 9436, Level 16, State 1, Line 1
XML parsing: line 1, character 18, end tag does not match start tag
```

XML stĺpec môže obsahovať iba „well-formed" XML podľa špecifikácie XML 1.0. Môže obsahovať dokumenty alebo ich časti.

Pri výpise pomocou príkazu SELECT sa výsledok zobrazí v tvare:

```
SELECT * FROM tblXml_tabulka;

id          xml_stlpec
---------   ----------
1           <doc />
```

Prípadne môžeme použiť explicitnú konverziu pomocou príkazu CAST:

```
SELECT CAST(xml_stlpec AS VARCHAR(MAX))FROM tblXml_tabulka
```

Stĺpec údajového typu nemôže byť primárnym kľúčom. Pokus o vytvorenie takéhoto stĺpca sa skončí chybou:

```
CREATE TABLE tblXml_tabulka
(
  xml_ stlpec XML PRIMARY KEY)

Msg 1919, Level 16, State 1, Line 1
Column 'xml_stlpec' in table ' tblXml_tabulka ' is of a type that is invalid for use
as a key column in an index.
```

V XML stĺpci môžeme definovať aj DEFAULT hodnoty, napríklad:

```
create table tblFaktury
(
  id INT,
  dokument XML DEFAULT '<invoice/>')
```

Potom pri vkladaní použijeme kľúčové slovo DEFAULT:

```
INSERT tblFaktury VALUES (1, DEFAULT)
```

Už z doterajších pokusov je zrejmé, že XML stĺpec nie je len bežný textový stĺpec, v pozadí je podporovaná široká paleta XML technológií. Obsah stĺpca môže byť validovaný voči XML Schéme, podporuje XML indexy, XQuery a XPath 2.0. Obsah stĺpca je konzistentný s XML funkcionalitou databázy FOR XML a OpenXML.

SQL server dokáže pracovať aj s XML schémou. Vytvorme jednoduchú databázovú tabuľku objednávok, pričom obsah (položky) objednávky bude uložený v údajovom type XML. Najskôr však vytvorme XML schému. XML schéma opisuje usporiadanie súčastí XML dokumentu a definuje ich typy. XML schémy môžu byť asociované s XML typom. Pomocou nich verifikujeme či údaje vložené do stĺpca údajového typu XML sedia so schémou. Súbor schém označujeme ako „schema collection".
Aj schéma XML dokumentu, ktorá definuje jeho typy, je XML dokumentom. V odporúčaní W3C pre XML sú špecifikované pravidlá definovania typov dokumentu. Pridružením deklarácie typov k XML dokumentu sa dosiahne možnosť uskutočniť kontrolu dokumentu. Dobre sformátovaný XML dokument, ktorý má pridruženú deklaráciu typu dokumentu a vyhovuje všetkým v nej definovaným obmedzeniam, je platný (valid). XML schému vytvoríme príkazom:

```
CREATE XML SCHEMA COLLECTION ObsahObjednavkySchema AS
'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <!-- deklaracia schémy -->

</xs:schema>'
```

Teraz môžeme vytvoriť tabuľku s XML datatypom a priradenou schémou:

```
CREATE TABLE tblObjednavky
(
  id integer PRIMARY KEY,
  datum datetime,
  ZakaznikID integer,
  ObsahObjednavky xml(ObsahObjednavkySchema))
```

XML funkcionalita bola významne rozšírená vo verzii 2008, hlavne v oblasti možnosti validácie XML dokumentu podľa schémy najmä o podporu validácie dátumu a času, bola rozšírená podpora XQuery a možnosti vkladania XML dokumentov do databázových tabuliek.

Validácia podľa XML schémy dokáže pracovať s takzvanými žolíkmi (wildcards)n pomocou deklarácie **any, anyAttribute** a **anyType**:

```
<xs:complexType name="Order" mixed="true">
  <xs:sequence>
    <xs:element name="CustomerName"/>
    <xs:element name="OrderTotal"/>
    <xs:any namespace="##other" processContents="skip"
minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Podľa takejto schémy môžeme validovať aj XML dokumenty, ktoré obsahujú aj iné atribúty ako CustomerName a OrderTotal. V našom príklade pribudol atribút shp:Delivery:

```
<Invoice xmlns="http://adventure-works.com/order"
       xmlns:shp="http://adventure-works.com/shipping">
  <Order>
    <CustomerName>Graeme Malcolm</CustomerName>
    <OrderTotal>299.99</OrderTotal>
    <shp:Delivery>Express</shp:Delivery>
  </Order>
</Invoice>
```

Môžeme validovať aj podľa vymenovaného zoznamu hodnôt. Napríklad ak máme definovaný zoznam veľkostí oblečenia:

```
xs:simpleType name="sizeListType">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="S"/>
        <xs:enumeration value="M"/>
        <xs:enumeration value="L"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
```

Môžeme deklarovať prípustné hodnoty:

```
<AvailableSizes>S M L</AvailableSizes>
```

Pre ilustráciu uvedieme aj zložitejší príklad validácie. Máme jednak sortiment oblečenia s písmenovým veľkostným označením ale aj sortiment bicyklov, kde veľkosť udáva číslo – rozmer rámu:

```
CREATE XML SCHEMA COLLECTION CatalogSizeSchema AS
N'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
       <xs:simpleType name="productSizeType">
              <xs:union>
                     <xs:simpleType>
                      <xs:list>
                            <xs:simpleType>
                             <xs:restriction base="xs:integer">
                                   <xs:enumeration value="18"/>
                                   <xs:enumeration value="20"/>
                                   <xs:enumeration value="22"/>
                                   <xs:enumeration value="24"/>
                             </xs:restriction>
                            </xs:simpleType>
                      </xs:list>
                     </xs:simpleType>
                     <xs:simpleType>
                      <xs:list>
                            <xs:simpleType>
                             <xs:restriction base="xs:string">
                                   <xs:enumeration value="S"/>
                                   <xs:enumeration value="M"/>
                                   <xs:enumeration value="L"/>
                             </xs:restriction>
                            </xs:simpleType>
                      </xs:list>
                     </xs:simpleType>
              </xs:union>
       </xs:simpleType>
</xs:schema>'
```

Aj v takomto prípade môžeme deklarovať prípustné hodnoty pre jedného odberateľa veľkosti oblečenia a pre iného veľkosti bicyklových rámov:

```
<Catalog>
  <Product>
    <ProductName>Road Bike</ProductName>
    <AvailableSizes>22 24</AvailableSizes>
  </Product>
  <Product>
    <ProductName>Cycling Jersey</ProductName>
    <AvailableSizes>S M L</AvailableSizes>
  </Product>
</Catalog>
```

Nad XML stĺpcami môžeme vytvoriť špeciálne XML indexy, ktoré optimalizujú XML požiadavky nad príslušným stĺpcom. Tabuľka alebo pohľad, v ktorej indexujeme XML údajový typ musí mať primárny kľúč typu „clustered". Najskôr musí byť vytvorený primárny XML index. Kompozitný XML index nie je podporovaný:

```
CREATE TABLE xml_tab
(
  id integer primary key,
  doc xml)
GO
CREATE PRIMARY XML INDEX xml_idx on xml_tab (doc)
GO
```

SQL Server 2005 podporuje 3 špecializované typy XML indexov: VALUE, PATH a PROPERTY.

XQuery

Klasický databázový jazyk SQL nie je pre dopytovanie vo vnútri XML údajových typov príliš vhodný. Pre tento účel je v SQL Serveri 2005 implementovaný jazyk XQuery. Jeho základom sú metódy XML typu

- xml.exist
- xml.value
- xml.query
- xml.modify
- xml.nodes

spolu v kombinácii s FLOWR príkazmi a konštrukciami. FLOWR je akronym od hlavných kľúčových slov XQurey výrazov: FOR, LET, WHERE, ORDER BY a RETURN Ak sa tieto XQuery metódy použijú s SQL príkazom SELECT môžu vracať stĺpce v „rowset-och".

FLOWR príkazy:

Príkaz	Popis
For	**Iteruje cez „súrodenecké" uzly.**
Where	**Filtrovacie kritérium pre iteráciu.**
Return	**Špecifikuje návratové XML.**

XQuery môžeme vyskúšať na údajoch v dátovom type XML, buď v tabuľke, alebo pre jednoduché pokusy nám postačí aj dočasná premenná dátového typu XML.

```
declare @x xml
set @x=
'<Faktury>
<Faktura>
            <Zakaznik>Jan Novak</Zakaznik>
            <Polozky>
                  <Polozka ProdID="2" Cena="1.99" Kusy="1" />
                  <Polozka ProdID="3" Cena="2.99" Kusy="2" />
                  <Polozka ProdID="5" Cena="1.99" Kusy="1" />
            </Polozky>
</Faktura>
      <Faktura>
            <Zakaznik>Zuzana Kratka</Zakaznik>
            <Polozky>
                  <Polozka ProdID="2" Cena="1.99" Kusy="1"/>
            </Polozky>
      </Faktura>
</Faktury>'
SELECT @x.query(
'<Zakaznici>
{
for $faktura in /Faktury/Faktura
return $faktura/Zakaznik}
</Zakaznici>')
```

V dopyte sme chceli zistiť zoznam zákazníkov. Výsledkom je XML dokument:

```
<Zakaznici>
  <Zakaznik>Jan Novak</Zakaznik>
  <Zakaznik>Zuzana Kratka</Zakaznik>
</Zakaznici>
```

Ak chceme do XML dokumentu vypísať všetky faktúry, sformulujeme príkaz:

```
SELECT @x.query('/Faktury/Faktura') AS objednane

<Faktura>
  <Zakaznik>Jan Novak</Zakaznik>
  <Polozky>
    <Polozka ProdID="2" Cena="1.99" Kusy="1" />
    <Polozka ProdID="3" Cena="2.99" Kusy="2" />
    <Polozka ProdID="5" Cena="1.99" Kusy="1" />
  </Polozky>
</Faktura>
<Faktura>
  <Zakaznik>Zuzana Kratka</Zakaznik>
  <Polozky>
    <Polozka ProdID="2" Cena="1.99" Kusy="1" />
  </Polozky>
</Faktura>
```

Metóda **xml.exists** vráti, či daný element v stĺpci s XML údajovým typom existuje alebo nie. V prípade existencie elementu vráti príkaz hodnotu 1, v opačnom prípade hodnotu 0.

Metóda **xml.value** nám vráti, hodnotu príslušného prvku. Prvok je určený indexmi v hranatých zátvorkách.

Pomocou metódy **xml.nodes** vyberáme príslušné uzly. Môžeme použiť klauzulu CROSS APPLY alebo OUTER APPLY. V prvom prípade použijeme klauzulu CROSS APPLY. Zobrazia sa len záznamy obsahujúce dokumenty s jedným elementom. Ak použijeme klauzulu OUTER APPLY, zobrazia sa všetky elementy.

SQL Server 2008 podporuje klauzulu **let** pre priradenie hodnôt do premenných. Ukážeme to na príklade:

```
declare @x xml
set @x=
'<Faktury>
<Faktura>
              <Zakaznik>Jan Novak</Zakaznik>
              <Polozky>
                     <Polozka ProdID="2" Cena="1.99" Kusy="1" />
                     <Polozka ProdID="3" Cena="2.99" Kusy="2" />
                     <Polozka ProdID="5" Cena="1.99" Kusy="1" />
              </Polozky>
</Faktura>
       <Faktura>
              <Zakaznik>Zuzana Kratka</Zakaznik>
              <Polozky>
                     <Polozka ProdID="2" Cena="1.99" Kusy="1"/>
              </Polozky>
       </Faktura>
</Faktury>'
SELECT @x.query(
'<Objednavky>
{
for $faktura in /Faktury/Faktura
let $count:=count($faktura/Polozky/Polozka)
order by $count
return
<Objednavka>
{$faktura/Zakaznik}
<Pocet>{$count}</Pocet>
</Objednavka>}
</Objednavky>')
```

Výsledkom je XML dokument:

```
Objednavky>
  <Objednavka>
    <Zakaznik>Zuzana Kratka</Zakaznik>
    <Pocet>1</Pocet>
  </Objednavka>
  <Objednavka>
    <Zakaznik>Jan Novak</Zakaznik>
    <Pocet>3</Pocet>
  </Objednavka>
</Objednavky>
```

Kapitola 12: Spatial – práca s geometrickými a geografickými údajmi

Technológia Spatial nám umožňuje prácu s viacrozmernými údajmi, napríklad bodmi, vektormi, mnohouholníkmi a rastrovými obrázkami, presnejšie s ich matematickou reprezentáciou uloženou v databáze. Môže ísť o dvojrozmerné úlohy typu: Je mesto XY v spádovej oblasti distribučnej firmy Z? Aké sú ceny pozemkov v okruhu dva kilometre od miesta kde chceme vybudovať priemyselný objekt, prípadne úlohy priestorové. Údaje o n rozmerných objektoch (body, čiary, mnohouholníky, kruhy, vektory...) sú v databáze uložené vo forme súradníc. Spatial dokáže spolupracovať s údajmi získanými z programov typu GIS (Geografické informačné systémy). SQL Server 2008 podporuje dvojrozmerné vektorové údaje.

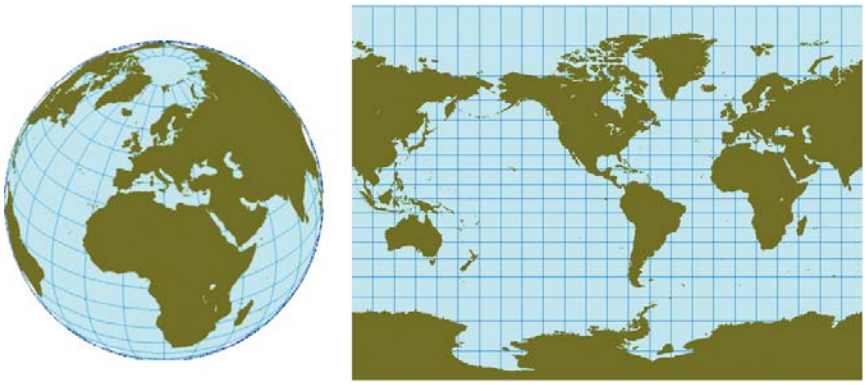
Ak sa zamyslíme nad princípom dopytovacieho databázového jazyka SQL, je to vlastne veľmi zjednodušená angličtina a keďže ide o príkazy, ktoré má databázový server vykonať, využíva vlastne len jediné gramatické pravidlo – rozkazovací spôsob. Ak by sme chceli do jazyka SQL sformulovať na prvý pohľad jednoduchú geografickú úlohu: „Ktoré cesty pretínajú areál spoločnosti Microsoft v Redmnonde" sformulovali by sme dopyt asi takto:

```
SELECT *
FROM roads
WHERE roads.geom.Intersects(@ms)=1
```

Geografická úloha typu „Ktorá cesta pretína určenú oblasť"

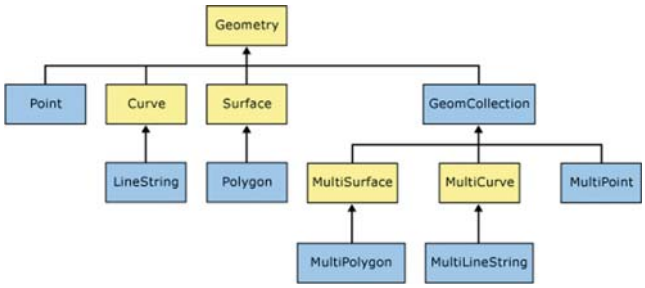
Geometrické a geografické údaje

Spatial na platforme servera SQL Server 2008 využíva dva typy údajov.
Geometrické údaje sú v planárnej alebo Euklidovskej (flat-earth) súradnicovej sústave definovanej konzorciom OGC (Open Geospatial Consortium).
Geografické údaje sú brané z elipsoindého (round-earth) modelu, čiže využívajú klasické súradnice – zemepisnú dĺžku a šírku, podobne ako GPS systémy.



Premietnutie „guľových" zemepisných súradníc do planárneho systému

Pre riešenie „spatial" úloh potrebujeme vytvoriť vhodnú inštanciu geometrie. Pracuje sa s jednoduchými geometrickými útvarmi a obrazcami ako sú bod, priamka, reťazec priamok a polygón, teda oblasť geometricky reprezentovaná ako ľubovoľný mnohouholník:



Spatial Geometry

Najskôr ukážeme definovanie základných objektov:

Point

```
declare @g geometry
set @g = geometry::Parse('POINT(5.5 8.8)')
```

LineString

```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('LINESTRING(1 1, 2 4, 3 9)', 0);
```

Polygon

```
DECLARE @g geometry;
SET @g = geometry::STPolyFromText('POLYGON((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 1,
1 1))', 10);
```

GeometryCollection

```
DECLARE @g geometry;
SET @g = geometry::STGeomCollFromText('GEOMETRYCOLLECTION(POINT(3 3 1), POLYGON((0 0
2, 1 10 3, 1 0 4, 0 0 2)))', 1);
```

Pre najjednoduchšie „spatial" úlohy nad údajmi v databázových tabuľkách je potrebné vytvoriť databázovú tabuľku s dvomi atribútmi, identifikátorom a údajovým typom „geometry". Pri reálnych úlohách použijeme aj ďalšie atribúty, ako sú názvy objektov, ceny a podobne. Ukážeme príklady pre geometrické aj geografické údaje.

Ukladanie geometrických objektov do spatial tabuľky

```
CREATE TABLE SpatialTable
    ( id int IDENTITY (1,1),
    GeomCol1 geometry,
    GeomCol2 AS GeomCol1.STAsText());
GO

INSERT INTO SpatialTable (GeomCol1)
VALUES (geometry::STGeomFromText('LINESTRING (100 100, 20 180, 180 180)', 0));


INSERT INTO SpatialTable (GeomCol1)
VALUES (geometry::STGeomFromText('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));

DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;

SELECT @geom1 = GeomCol1 FROM SpatialTable WHERE id = 1;
SELECT @geom2 = GeomCol1 FROM SpatialTable WHERE id = 2;
SELECT @result = @geom1.STIntersection(@geom2);
SELECT @result.STAsText();
```

SQL server nám vráti výsledok dopytu v tvare

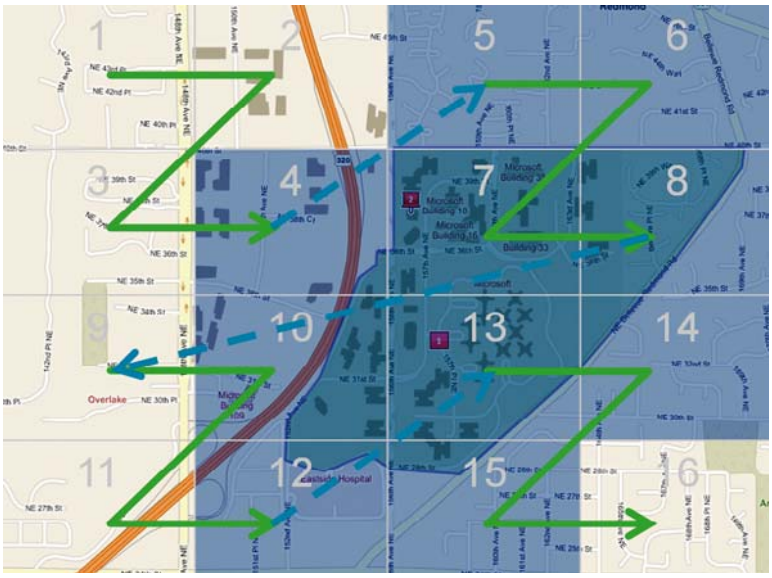
```
WKT
------------------------------------
LINESTRING (50 150, 100 100)
```

Ukladanie **geografických** objektov do spatial tabuľky:

```
CREATE TABLE SpatialTable
    ( id int IDENTITY (1,1),
    GeogCol1 geography,
    GeogCol2 AS GeogCol1.STAsText());
GO

INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('LINESTRING(47.656 -122.360, 47.656 -122.343)',
4326));

INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('POLYGON((47.653 -122.358, 47.649 -122.348, 47.658
-122.348, 47.658 -122.358, 47.653 -122.358))', 4326));

DECLARE @geog1 geography;
DECLARE @geog2 geography;
DECLARE @result geography;

SELECT @geog1 = GeogCol1 FROM SpatialTable WHERE id = 1;
SELECT @geog2 = GeogCol1 FROM SpatialTable WHERE id = 2;
SELECT @result = @geog1.STIntersection(@geog2);
SELECT @result.STAsText();
```

Spatial indexy

Pre efektívne riešenie zložitejších úloh je potrebné údaje v databázových tabuľkách indexovať. Spatial indexy využívajú viacúrovňový tabuľkový systém s rôznou veľkosťou mriežky, pričom každé pole mriežky je možné rozdeliť na menšie políčka s rozlíšením LOW: 4x4, MEDIUM: 8x8 alebo HIGH: 16x16. Takýto systém umožňuje oveľa vyššiu flexibilitu ako jednoduchá mriežka. Príklad indexovania mriežky je na obrázku.

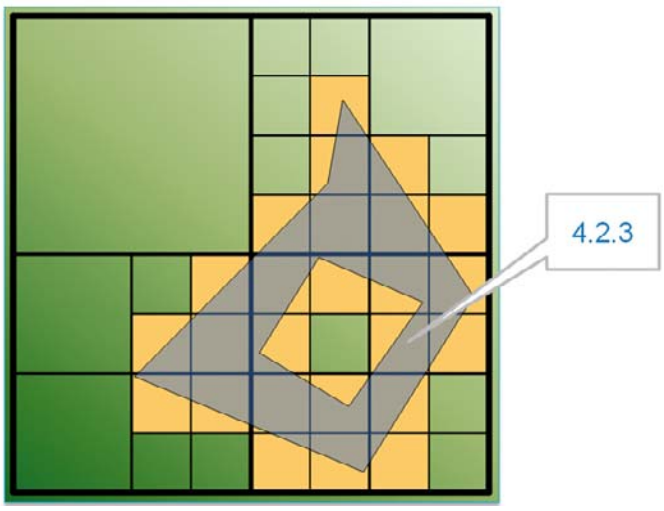


Princíp indexovania geografických údajov

Spatial index vytvoríme pomocou príkazu:

```
CREATE SPATIAL INDEX sixd
ON Tspatial(region)
WITH (
       BOUNDING_BOX = (0, 0, 500, 500),
       GRIDS = (LOW, LOW, MEDIUM, HIGH),
       CELLS_PER_OBJECT = 20)
```

Ďalší obrázok ukazuje princíp rozdelenia multi-level mriežky na menšie oblasti a príklad adresovania poľa.

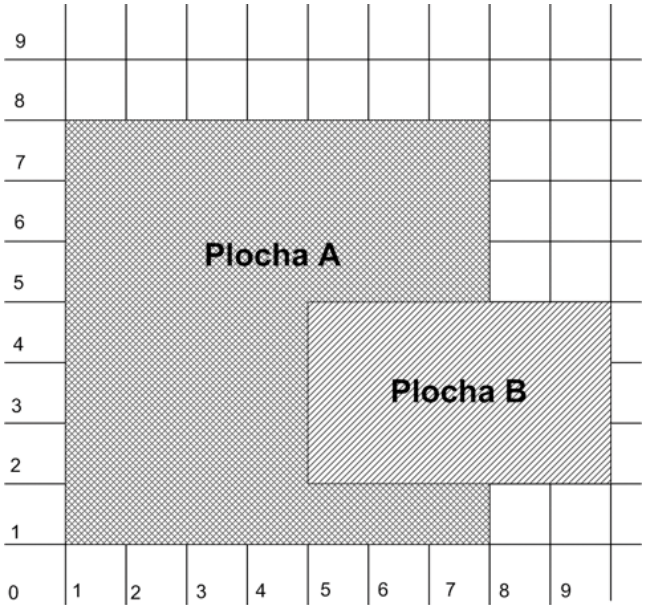


Multi-level grid

Technológiu Spatial najlepšie pochopíme na niekoľkých názorných príkladoch.

Príklad pre geometrické objekty

Majme rovinu, ktorá obsahuje dva plošné geometrické útvary. Tieto útvary sú zadefinované pomocou súradnicového systému s počiatkom (0,0) v ľavom dolnom rohu. Môžu byť grafickou reprezentáciou čohokoľvek, napríklad mapy pozemkov, geografickej sféry marketingového záujmu...



Definícia plôch pre cvičný príklad

Vytvorme tabuľku PLOCHY, ktorá bude obsahovať okrem primárneho kľúča ešte meno a ohodnotenie plochy. Pod pojmom ohodnotenie by sme mohli chápať napríklad cenu pozemku, percento vplyvu a podobne.

```
CREATE TABLE Plochy
(
    id int PRIMARY KEY,
    plocha geometry, );
```

Teraz môžeme prikročiť k definovaniu nami použitých plôch:

Plocha A je štvorec. Štvorec môžeme jednoznačne popísať súradnicami jeho vrcholov, teda v našom prípade `(1 1, 8 1, 8 8, 1 8, 1 1)`:

```
INSERT INTO Plochy
VALUES (1, geometry::STGeomFromText('POLYGON ((1 1, 8 1, 8 8, 1 8, 1 1))', 0));
```

Ak zadáme nesprávne súradnice, a polygón nie je uzavretý, napríklad preto, že prvý a posledný bod nie sú totožné, vypíše sa chybové hlásenie:

```
VISTAVYVOJ-PC(VistaVyvoj-PC\VistaVyvoj): Msg 6522, Level 16, State 1, Line 1
A .NET Framework error occurred during execution of user defined routine or aggregate
'geometry':
System.FormatException: 24119: The Polygon input is not valid because the start and
end points of the exterior ring are not the same. Each ring of a polygon must have the
same start and end points.
...
The statement has been terminated.
```

Plocha B je obdĺžnik. Podobne ako štvorec ho môžeme jednoznačne popísať súradnicami jeho vrcholov:

```
INSERT INTO Plochy
VALUES (2, geometry::STGeomFromText('POLYGON ((5 2, 10 2, 10 5, 5 5, 5 2))', 0));
```

V tomto okamihu teda máme v tabuľke plochy uloženú našu schému pozostávajúcu z dvoch obrazcov. Nad týmito údajmi môžeme riešiť mnoho jednoduchých aj pomerne zložitých typov úloh, ktoré napríklad poznáme zo školskej geometrie.

Môžeme napríklad zistiť, či existuje prienik dvoch zadaných plôch A a B, ktoré podľa obrázka očividne prienik obsahujú. Pre tento účel využijeme operáciu **STIntersection**:

```
DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;

SELECT @geom1 = plocha FROM Plochy WHERE id = 1;
SELECT @geom2 = plocha FROM Plochy WHERE id = 2;
SELECT @result = @geom1.STIntersection(@geom2);
SELECT @result.STAsText();
```

Výsledkom (prienikom) je polygón so súradnicami:

```
POLYGON ((5 2, 8 2, 8 5, 5 5, 5 2))
```

Ako prienik sme získali popis geometrie obrazca `(5,5, 5,2, 8,2, 8,5, 5,5)`.

Môžeme vypočítať plochu tohto prieniku pomocou metódy STArea:

```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('POLYGON((5 2, 8 2, 8 5, 5 5, 5 2))', 0);
SELECT @g.STArea();
```

```
9
```

Podobne ako prienik môžeme riešiť zjednotenie, teda celkovú plochu, ktorú predstavujú dve v našom prípade prekrývajúce sa plochy A a B. Pre tento účel využijeme operáciu **STUnion**:

```
DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;

SELECT @geom1 = plocha FROM Plochy WHERE id = 1;
SELECT @geom2 = plocha FROM Plochy WHERE id = 2;
SELECT @result = @geom1.STUnion(@geom2);
SELECT @result.STAsText();
```

Výsledkom je polygón so súradnicami:

```
POLYGON ((1 1, 8 1, 8 2, 10 2, 10 5, 8 5, 8 8, 1 8, 1 1))
```

Príklad určovania vzdialeností v geografických súradniciach

Praktické príklady hlavne z oblasti geografických biznis analýz sú vo väčšine prípadov vztiahnuté na zemepisné súradnice a riešia sa úlohy pokrytia zákazníkov a podobne.

Námetom príkladu bude situácia, kedy firma má v rámci efektívneho pokrytia globálneho ekonomického priestoru niekoľko zásobovacích skladov. Riešime typickú úlohu, pre každú pobočku určiť z ktorého skladu je najvýhodnejšie túto pobočku zásobovať, alebo exaktne povedané, na základe zemepisných súradníc zásobovacích skladov a pobočiek sa potrebujeme rozhodnúť, od ktorého skladu je najbližšie k príslušnej pobočke. V našom konkrétnom príklade má firma v Európe dva zásobovacie sklady. Jeden v Mníchove a druhý v Ríme. Úlohu budeme riešiť pre dve pobočky. Prvá z nich sídli v Prahe a druhá v Benátkach. Zemepisné súradnice uvedených miest sú:

```
Praha     14.41935,  50.09193
Rím       12.4833,   41.9601
Mníchov   11.5424,   48.2231
Benátky   12.28389,  45.45233
```

Východisková geografická situácia pre námet cvičného príkladu

Vytvoríme tabuľku skladov:

```
CREATE TABLE Sklady
(
   id int PRIMARY KEY,
   mesto varchar(20),
   suradnice geography );
```

a naplníme údajmi pre mestá, pre ktoré budeme riešiť príklad:

```
INSERT INTO Sklady
VALUES (1, 'Rim', geography::STGeomFromText('POINT(12.4833 41.9601)', 4326));

INSERT INTO Sklady
VALUES (2, 'Mnichov', geography::STGeomFromText('POINT(11.5424 48.2231)', 4326));
```

Podobne vytvoríme tabuľku pobočiek. Pre jednoduchosť má úplne rovnakú štruktúru ako tabuľka skladov:

```
CREATE TABLE Pobocky
(
   id int PRIMARY KEY,
   mesto varchar(20),
   suradnice geography );


INSERT INTO Pobocky
VALUES (1, 'Praha', geography::STGeomFromText('POINT(14.41935 50.09193)', 4326));


INSERT INTO Pobocky
VALUES (2, 'Benatky', geography::STGeomFromText('POINT(12.28389 45.45233)', 4326));
```

Pričom súradnice `14.41935 50.09193` v prvom zázname určujú zemepisnú dĺžku a šírku Prahy. Pozornosť si zaslúži vysvetlenie významu parametra SRID, ktorý udáva druh zemepisných súradníc. Konštanta 4326 je pre systém zemepisných súradníc WGS 84, teda bežne používaný a zo školských atlasov dobre známy systém uhlových mier zemepisných dĺžok a šírok.

Po vytvorení a naplnení cvičných tabuliek a prípravných úkonoch, môžeme začať riešiť geografické úlohy. Ako prvú úlohu zistíme vzdialenosti pobočiek od skladov:

```
SELECT P.mesto AS 'Pobocka', S.mesto AS 'Sklad',
 P.suradnice.STDistance(S.suradnice)
FROM Sklady S, Pobocky P


S Pobocka            Sklad
-------------------- -------------------- ----------------------
Praha                Rim                  906148,58340179
Praha                Mnichov              377360,123348305
Benatky              Rim                  380404,107941765
Benatky              Mnichov              312783,529829918
```

Vzdialenosti vo výslednej tabuľke sú v metroch. V druhej úlohe budeme zisťovať, z ktorého skladu je najvýhodnejšie zásobovať ktorého zákazníka. Intuitívne by sme odpovedali, že zákazníka z Benátok budeme zásobovať z Ríma a zákazníka z Prahy z Mníchovského skladu. Ak sa však pozrieme na mapu, zistíme, že do Benátok je bližšie z Mníchova než z Ríma:

```
SELECT S.mesto AS 'Sklad', P.suradnice.STDistance(S.suradnice)
FROM Sklady S, Pobocky P
WHERE P.mesto = 'Benatky'
AND P.suradnice.STDistance(S.suradnice) =
(SELECT MIN(P1.suradnice.STDistance(S1.suradnice)) FROM Sklady S1, Pobocky P1)


Sklad
-------------------- ----------------------
Mnichov              312783,529829918
```

Príklad určovania príslušnosti k lokalite

Častým okruhom úloh je určovanie príslušnosti geometrického, alebo geografického objektu do nejakej oblasti. Typickým príkladom je úloha určiť do ktorej mestskej časti patrí zadaná ulica. Pre túto úlohu vytvoríme a naplníme dve tabuľky. V jednej budú údaje o mestských častiach fiktívneho mesta a v druhej údaje o uliciach. Pre jednoduchosť budeme úlohu riešiť v geometrických súradniciach na lineárnom pláne mesta. Mestské časti budú ohraničené polygónom a ulice budú tvorené ako postupnosť vektorov, teda LINESTRING:

```
CREATE TABLE MestskeCasti
(
   CastId int IDENTITY (1,1),
   CastNazov nvarchar(20),
   CastGeo geometry);

INSERT INTO MestskeCasti (CastNazov, CastGeo)
VALUES ('Stare mesto',
geometry::STGeomFromText('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));


INSERT INTO MestskeCasti (CastNazov, CastGeo)
VALUES ('Slnecne namestie',
geometry::STGeomFromText('POLYGON ((300 0, 150 0, 150 150, 300 150, 300 0))', 0));


INSERT INTO MestskeCasti (CastNazov, CastGeo)
VALUES ('Mestsky park',
geometry::STGeomFromText('POLYGON ((150 0, 300 0, 300 300, 150 300, 150 0))', 0));


CREATE TABLE Ulice
(      UlicaId int IDENTITY (1,1),
       UlicaMeno nvarchar(20),
    UlicaGeo geometry);

INSERT INTO Ulice (UlicaMeno, UlicaGeo)
VALUES ('Hlavna ulica',
geometry::STGeomFromText ('LINESTRING (100 100, 20 180, 180 180)', 0))

INSERT INTO Ulice (UlicaMeno, UlicaGeo)
VALUES ('Lipova alej',
geometry::STGeomFromText ('LINESTRING (300 300, 300 150, 50 50)', 0))
```

Teraz môžeme riešiť úlohu, cez ktoré mestské štvrte prechádzajú zadané ulice:

```
SELECT UlicaMeno, CastNazov
FROM MestskeCasti c, Ulice u
WHERE u.UlicaGeo.STIntersects(CastGeo) = 1
ORDER BY UlicaMeno

UlicaMeno            CastNazov
-------------------- --------------------
Hlavna ulica         Stare mesto
Hlavna ulica         Mestsky park
Lipova alej          Stare mesto
Lipova alej          Slnecne namestie
Lipova alej          Mestsky park
```

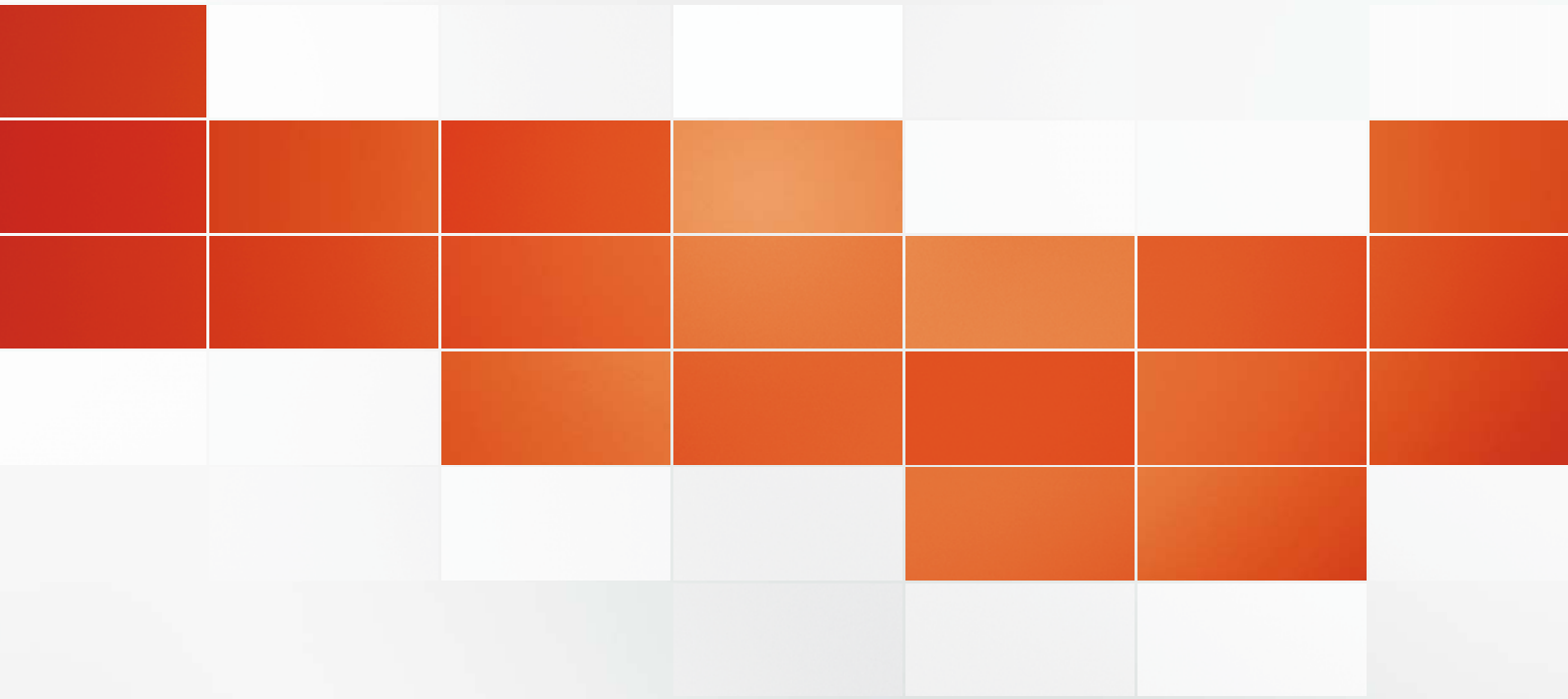
Príloha: Inštalácia cvičných databáz

Cvičné databázy AdventureWorks2008 a AdventureWorks2008 DW BI môžeme stiahnuť na adrese http:// www.codeplex.com. Inštalačné programy jednotlivých verzií cvičnej databázy AdventureWorks2008 majú približne 50 MB a umožnujú jej nainštalovanie na lokálny server. Po spustení inštalačného programu sa vytvorí adresár.

c:\Program Files\Microsoft SQL Server\100\Tools\Samples\

V tomto adresári je cvičná databáza uložená vo forme záložného súboru s príponou BAK. Skript pre obnovu tejto databázy je v súbore RestoreAdventureWorks2008xx.sql.
Tento skript spustíme pomocou nástroja SQL Server Management Studio. Skript obsahuje v komentári priame nastavenie adresára, v ktorom sa nachádza súbor zálohy. Aby sme sa vyhli problémom s prideľovaním prístupových práv do podadresárov adresára c:\Program Files, prekopírovali sme súbor zálohy do novovytvoreného adresára c:\Install, ktorý má štandardné prístupové práva.

```
USE master;
DECLARE @source_path nvarchar(256);
SET @source_path = 'c:\Install\'
BEGIN
       DECLARE @sql_path nvarchar(256);
       SELECT @sql_path = SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf',
    LOWER(physical_name)) - 1)
       FROM master.sys.master_files WHERE database_id = 1 AND file_id = 1;
       IF EXISTS (SELECT * FROM sys.databases
        WHERE name = 'AdventureWorksLT2008')
       BEGIN
          EXECUTE (N'ALTER DATABASE AdventureWorksLT2008
                    SET SINGLE_USER WITH ROLLBACK IMMEDIATE;');
          EXECUTE (N'DROP DATABASE AdventureWorksLT2008;');
        END
        EXECUTE (N'RESTORE DATABASE AdventureWorksLT2008
        FROM DISK = ''' + @source_path +
     'Tools\Samples\AdventureWorksLT2008.bak''
        WITH
        MOVE ''AdventureWorksLT2008_Data'' TO N''' + @sql_path +
       N'AdventureWorksLT2008.mdf'',
         MOVE ''AdventureWorksLT2008_Log''  TO N''' + @sql_path +
      N'AdventureWorksLT2008.ldf'';');
       EXECUTE (N'ALTER DATABASE AdventureWorksLT2008 SET NEW_BROKER;');
END
```



www.microsoft.com/slovakia/sqlserver