ELSŐ FEJEZET

RDBMS alapok: miből áll az SQL Server adatbázis?

Miből áll egy adatbázis? Természetesen adatokból. (Mit ér az olyan adatbázis, amely nem tárol semmit sem?) De egy *relációs adatbázis-kezelő rendszer* (*RDBMS*) többet jelent egyszerű adatoknál. Manapság a fejlett RDBMS-ek nemcsak tárolják az adatokat; hanem kezelik is, valamint korlátozzák, hogy az adatbázisba milyen adatok kerülhetnek be, és segítik az adatok kinyerését is a rendszerből. Ha csupán annyi a célunk, hogy az adatokat valahová beömlesszük, akkor bármilyen adattárolási rendszert használhatunk. Az RDBMS az adattároláson túl azt a lehetőséget is biztosítja, hogy meghatározzuk az adatok megjelenését, illetve az adatokra vonatkozó üzleti szabályokat.

Ne keverjük össze az általam „az adatok üzleti szabályainak" nevezett szabályokat az általánosabb érvényű működési szabályokkal, amelyek az egész rendszer munkáját irányítják (például a felhasználó addig nem lát semmit, amíg be nem lép a rendszerbe, vagy automatikusan nem állítja be a könyvelőrendszerben az aktuális periódust a hónap első napján). Ezek a szabályok a rendszer bármely szintjén érvényesíthetők. (Manapság, ezeket a szabályokat az n rétegű rendszerek középső vagy kliensrétegében találhatjuk.) A működési szabályok helyett itt az adatokra vonatkozó üzleti szabályokról van szó. Például egy megrendelés nem rendelkezhet negatív összeggel. RDBMS esetén ezeket a szabályokat adatbázisszinten lehet megfogalmazni.

Ez a fejezet áttekintést ad a könyv többi részéről. A fejezetben ismertetett témákkal a későbbi fejezetek is foglalkoznak, de ez a fejezet egyfajta fejlesztési tervet nyújt, amelyet a könyv olvasása során érdemes szem előtt tartani. Ez a fejezet részletesen bemutatja:

- az adatbázis-objektumokat,
- az adattípusokat,
- az adatintegritást biztosító egyéb adatbázis-koncepciókat.

Az adatbázis-objektumok áttekintése

Az RDBMS, mint például az SQL Server sok objektumot tartalmaz. Számos objektumorientált programozással foglalkozó szakember kritizálja, hogy mit tekint (vagy nem tekint) a Microsoft egy objektum definíciójának. Az SQL Server esetében a legfontosabb adatbázis objektumok a következők:

- Az adatbázis
- A tranzakciós napló
- Táblák
- Fájlcsoportok
- Diagrammok
- Nézetek
- Tárolt eljárások
- Felhasználói függvények
- Indexek
- Szerelvények
- Jelentések
- Teljesszövegű katalógusok
- Felhasználói adattípusok
- Szerepek
- Felhasználók

Az adatbázis-objektum

Az adatbázis a legmagasabb szintű objektum, amelyre az SQL Serverben hivatkozni lehet. (Technikailag maga a kiszolgáló is objektumnak tekinthető, de igazi „programozási" szempontból nem, ezért ezzel ne foglalkozzunk részletesebben.) A legtöbb, de nem az összes SQL Server objektum az adatbázis-objektum gyermeke.

> *Ha ismerjük az SQL Server korábbi verzióit, most azt kérdezhetjük: „Hogyan? Mi történt a bejelentkezésekkel? Mi történt a távoli kiszolgálókkal és az SQL-ügynök feladatokkal?" Az SQL Server több más objektummal is rendelkezik (mint azt a korábbi listában láttuk), amelyek célja az adatbázis támogatása. A csatolt kiszolgálók és talán az Integration Services csomagok kivételével ezek az adatbázis-rendszergazda hatáskörébe tartoznak, ezért általában nem vesztegetünk rájuk túl sok energiát a tervezési és programozási folyamatok során. (Ezek az SQL Management Objects [SMO] segítségével programozhatók, de ez annyira speciális eset, hogy ezzel most nem is foglalkozunk.)*

Az adatbázis tipikusan egy csoport, amely legalább a táblaobjektum halmazát, valamint gyakran más objektumokat is tartalmaz, például tárolt eljárásokat és nézeteket, amelyek az adatbázis tábláinak speciális adatcsoportosítását biztosítják.

Milyen táblatípusokat lehet egyetlen adatbázisban tárolni, és mi kerül külön adatbázisokba? Erre a kérdésre részletesebben visszatérünk a könyv későbbi részében, de most az egyszerűbb megközelítést választva azt mondhatjuk, hogy minden olyan adatot, amelyről általában úgy gondoljuk, hogy egy rendszerhez tartozik, vagy szorosan összekapcsolódik, egy adatbázis tárol. Az RDBMS, mint például az SQL Server, egyetlen kiszolgálón több felhasználói adatbázissal, illetve egyetlen adatbázissal is rendelkezhet. Az SQL Server által tárolható adatbázisok száma olyan tényezőktől függ mint a kapacitás (processzorteljesítmény, lemez I/O-korlátok, memória stb.), az autonómia (egyik felhasználó kezelési jogosultságokkal rendelkezik a kiszolgálón, ahol a rendszer fut, míg egy másik felhasználó rendszergazdajogokkal bír egy másik kiszolgálón), illetve az a tény, hogy a vállalat vagy az ügyfél hány adatbázissal rendelkezik. Több kiszolgáló egyetlen termelési adatbázissal rendelkezik; más kiszolgálók többel is. Ne feledjük, hogy az SQL Server pillanatnyilag rendelkezésre álló verzióiban (az SQL Server 2000 már öt éve a piacon volt, amikor a következő verzió felváltotta, tehát valószínűleg ez vagy a frissebb verzió kapható) több SQL Server példánnyal rendelkezhetünk, amelyek külön bejelentkezést és felügyeleti jogokat biztosítanak ugyanazon a fizikai kiszolgálón.

> *Valószínűleg sokakban felmerül a kérdés: ugyanazon a gépen futhat több, különböző SQL Server verzió, mondjuk az SQL Server 2000 és az SQL Server 2005 is? A válasz igen. Az SQL Server 2000 és 2005 használható ugyanazon a számítógépen. A magam részéről nem igazán bízom egy ilyen konfigurációban, még migrációs forgatókönyvek esetén sem, de ha szükség van rá, meg lehet csinálni.*

Amikor először töltjük be az SQL Servert, négy rendszeradatbázissal kezdhetjük a munkát:

- master,
- model,
- msdb,
- tempdb.

Ezeket az adatbázisokat a kiszolgáló helyes működésének érdekében telepíteni kell. (Valójában néhányuk nélkül a kiszolgáló teljesen működésképtelen.) Ezen túl a többi összetevő a választott telepítési paraméterektől függ. Néhány példa azok közül az adatbázisok közül, amelyekkel találkozhatunk:

- AdventureWorks (a mintaadatbázis),
- AdventureWorksDW (az Analysis Services mintaadatbázisa).

A rendszer által telepített példákon kívül a könyvben rengeteg régebbi példával is találkozhatunk (a példák telepítésével kapcsolatos információkért lásd az „F" függeléket):

- pubs,
- Northwind.

A könyv tervezése során rengeteg vita volt arról, hogy az újabb példák vagy a már bevált, jó öreg példák kerüljenek a könyvbe. Egészen őszinte leszek, amikor bevallom, hogy a Microsoft nem örült maradéktalanul a választásomnak, mely szerint ragaszkodtam a régi példákhoz, de ezért nem kérek bocsánatot.

Az új AdventureWorks adatbázis bizonyára robusztus példa, és remek példákkal szolgál minden apró részletről, amelyet az SQL Server 2005 segítségével kihasználhatunk. Azonban a probléma a példa összetettségében rejlik. Oktatási adatbázisként az AdventureWorks adatbázis túlságosan összetett. Olyan jellemzőket hoz példaként, amelyekkel csak kivételes esetekben találkozhatunk, és azokat domináns jellemzőként mutatja be. Több barátomat is megkérdeztem, akik tanítanak és/vagy írnak SQL Server anyagokat, és mindegyikük osztotta a véleményemet: bár a Northwind és a pubs több szempontból túl leegyszerűsített példák, nagyban megkönnyítik az SQL Server alapfogalmainak megértését. Jobban örülök annak, ha sikerül megértetni az alapokat, majd továbblépni, semmint az AdventureWorks felesleges összetettségével fárasztani az Olvasót.

A master adatbázis

A verziótól és az egyedi változtatásoktól függetlenül minden SQL Server rendelkezik master adatbázissal. Ez az adatbázis speciális táblákat (rendszertáblákat) tárol, amely a rendszer egészét követi nyomon. Például, amikor új adatbázist hozunk létre a kiszolgálón, a master adatbázis `sysdatabases` táblájába új bejegyzés kerül. A kiterjesztett és rendszer tárolt eljárásokat ez az adatbázis tárolja attól függetlenül, hogy melyik adatbázissal szeretnénk használni. Mivel szinte minden, a kiszolgálót leíró adatot itt tárol a rendszer, nyilvánvaló, hogy ez az adatbázis kulcsfontosságú a rendszer működésének szempontjából, és nem lehet törölni.

A rendszertáblák, a master adatbázis tábláit is beleértve, rendkívül hasznos objektumok. Segítségükkel meghatározhatjuk, hogy bizonyos objektumok léteznek-e, mielőtt azokkal bármilyen műveletet végeznénk. Például, ha olyan objektumot próbálunk meg létrehozni, amely egy adott adatbázisban már létezik, hibaüzenetet kapunk. Ha mégis erőltetni szeretnénk a dolgot, akkor érdemes megnézni, hogy a `sysobjects` tábla tartalmaz-e a táblára vonatkozó bejegyzést az adott adatbázisban. Ha igen, akkor az objektumot törölnünk kell, mielőtt újra létrehoznánk.

Ha a valaki túl könnyelmű, a „Remek, már alig várom, hogy a rendszertáblákkal játszhassak!" gondolat fogalmazódhat meg benne. *Az ilyen felhasználó ne menjen a rendszertáblák közelébe!* A rendszertáblák használata bármilyen formában igen kockázatos tevékenység. A Microsoft az SQL Server utolsó verziói közül legalább az utolsó három verziójában ellenzi a rendszertáblák használatát. A Microsoft semmiféle garanciát nem vállal a master adatbázis kompatibilitását illetően a különböző verziók között, csak azt garantálja, hogy azok változnak. A legrosszabb támadás a master adatbázis objektumainak módosítása során várható. Nem túlzok, amikor azt állítom, hogy ezeknek a tábláknak bármiféle módosítása egy működésképtelen SQL Servert eredményez. Szerencsére több lehetőség is rendelkezésünkre áll (például rendszerfüggvények, rendszer tárolt eljárások és információ_séma nézetek), amelyek segítségével a rendszertáblákban tárolt metaadatok nagy része kinyerhető.

Mindezek ellenére vannak helyzetek, ahonnan nincs visszaút. Látni fogunk olyan helyzeteket is, ahol nem tudjuk elkerülni a rendszertáblák használatát, de általában jobb ezekre a táblákra a szomszéd kannibál törzs tagjaiként gondolni, és békén hagyni őket.

A modell adatbázis

A modell adatbázis neve találó abból a szempontból, hogy olyan modellt jelent, amelyre nyugodtan alapozhatjuk a másolást. A modell adatbázis a létrehozni kívánt új adatbázisok számára biztosít egyfajta sablont. Ez azt jelenti, hogy szükség szerint módosíthatjuk a `modell` adatbázist, ha szeretnénk megváltoztatni a standard, újonnan létrehozott adatbázisok megjelenését. Például naplótáblákat adhatunk az adatbázishoz, amelyeket az összes jövőben készített adatbázisunkba beépíthetünk. Felhasználócsoportokat is beépíthetünk, amelyeket a rendszeren létrehozott összes új adatbázisba klónozhatunk. Ne feledjük, hogy mivel ez az adatbázis a többi adatbázis számára sablonként szolgál, erre az adatbázisra szükség van, és a rendszeren kell hagynunk; az adatbázis nem törölhető.

A modell adatbázis módosítása során több dologról nem szabad megfeledkeznünk. Először is bármely adatbázisnak, amelyet létre kell hoznunk, legalább akkora méretűnek kell lennie, mint amekkora a `modell` adatbázis. Ez azt jelenti, hogy ha a `modell` adatbázis méretét 100 MB-ra módosítjuk, akkor nem hozhatunk létre 100 MB-nál kisebb adatbázist. Több, ehhez hasonló csapda létezik, ezért a telepítések 90%-ánál ajánlom, hogy hagyjuk békén ezt az adatbázist.

Az msdb adatbázis

Az `msdb` az hely, ahol az SQL-ügynök a rendszerfeladatokat tárolja. Ha az adatbázis éjszakánkénti biztonsági mentését ütemezzük, erről bejegyzés készül az `msdb`-ben. A tárolt eljárást ütemezzük egyszeri végrehajtásra, és igen, ez a bejegyzés is bekerül az `msdb`-be.

A tempdb adatbázis

A `tempdb` a kiszolgáló egyik kulcsfontosságú munkaterülete. Ha összetett vagy nagy lekérdezést futtatunk, amelyhez az SQL Server köztes táblákat épít, ezeket a táblákat a `tempdb`-ben hozza létre. Amikor saját ideiglenes táblát hozunk létre, az a `tempdb`-be kerül, még akkor is, ha úgy gondoljuk, hogy azt az aktuális adatbázisban hozzuk létre. Ha felmerül az adatok ideiglenes tárolásának igénye, valószínű, hogy az adatokat a `tempdb`-ben tárolja a rendszer.

A `tempdb` eléggé különbözik a többi adatbázistól. Nemcsak azért, mert a tárolt objektumok ideiglenesek, hanem mert maga az adatbázis is ideiglenes. A `tempdb` megkülönböztető jegye, hogy a rendszerben az egyetlen adatbázis, amely az SQL Server indításával minden egyes alkalommal a semmiből épül újra.

Gyakorlatilag mi magunk is létrehozhatunk objektumokat a `tempdb`-ben, azonban én nagyon ellenzem ezt a gyakorlatot. Bármilyen adatbázisból, amelyhez hozzáférésünk van, létrehozhatunk ideiglenes objektumokat, azokat a `tempdb` fogja tárolni. Azzal, hogy az objektumokat közvetlenül a `tempdb`-ben hozzuk létre, nem nyerünk semmit, de növeljük az adatbázisokon keresztüli objektumhivatkozások keltette zűrzavart. Ez is egyike az „Inkább hagyjuk békén!" dolgoknak.

AdventureWorks

Az SQL Server már az AdventureWorks előtt is tartalmazott példákat. De a régi példáknak is voltak gyenge pontjaik. Például rossz minőségű tervezési gyakorlattal rendelkeztek. (Tartózkodom annak vitatásától, hogy az AdventureWorks esetén ugyanez a probléma vagy sem. Mondjuk, hogy az AdventureWorks egyebek mellett ezt a problémát is megpróbálta megoldani.) Ezen kívül egyszerűsített példák voltak, amelyek bizonyos adatbázis-koncepciók bemutatására összpontosítottak, nem pedig az SQL Serverre mint termékre vagy akár adatbázisra.

A Yukon (a ma SQL Server 2005 néven ismert termék belső fedőneve) korai fejlesztési szakaszaitól kezdve a Microsoft már tudta, hogy sokkal robusztusabb mintaadatbázist akar, amely a lehető legtöbb terméknél használható. Ennek az elképzelésnek az eredménye az AdventureWorks. Bár sokat fogok panaszkodni arra, hogy ez az adatbázis túlzottan komplex kezdő felhasználók számára, abban az értelemben valóban remekmű, hogy *mindent* megmutat. Oké, talán nem mindent, de egy majdnem teljes példa a valósághoz közel álló adatmennyiséggel, összetett struktúrákkal és olyan részekkel, amelyek a termék tulajdonságainak javát bemutatják. Ebben az értelemben igazán ragyogó példa.

Ahogyan a termék egyre fejlettebb tulajdonságokat tárgyaljuk, itt-ott használni fogom.

AdventureWorksDW

Ez az Analysis Services példája. (A DW adattárházat – datawarehouse – jelent, ami olyan adatbázistípus, amelyre a legtöbb Analysis Services projekt épül.) Ebben az a legnagyszerűbb dolog, hogy a Microsoft előrelátóan összekötötte a tranzakciós adatbázis példát az elemzési példával, és olyan példakészletet biztosít, amely bemutatja a kettő együttműködését is.

Ez a könyv nem foglalkozik a döntéstámogató adatbázisokkal, ezt az adatbázist nem is fogjuk használni, de ne feledkezzünk meg róla, ha elindítjuk az Analysis Services-t és kísérletezni kezdünk. Vizsgáljuk meg a két adatbázis közötti különbségeket. Ugyanazt a kitalált vállalatot szolgálják ki, de különböző céllal; tanuljunk belőlük.

A pubs adatbázis

Ah, a jó öreg `pubs`! Olyan, mint egy régi barát. A `pubs` csak a Microsoft webhelyéről külön letölthető példaként telepíthető, és elsősorban az oktató jellegű cikkek, és az ehhez hasonló könyvek segédanyaga. A `pubs`nak semmi köze az SQL Server működéséhez. A célja csupán annyi, hogy az oktatás és a kísérletezés számára konzisztens helyet biztosítson. A könyvben néhány helyen találkozni fogunk a `pubs` adatbázissal.

A `pubs` külön ugyan, de telepítető, és számottevő következmények nélkül törölhető is.

A Northwind-adatbázis

Ha korábbi programozási tapasztalataink kiterjedtek az Access vagy a Visual Basic programokra is, akkor már valószínűleg ismerjük a `Northwind`-adatbázist. A `Northwind` az SQL Server 7.0 újdonsága volt, de az SQL Server 2005 alapszintű telepítésének már nem része. A `pubs` adatbázishoz hasonlóan az SQL Server alapszintű telepítésétől külön kell telepítenünk. (Szerencsére része ugyanannak a letölthető csomagnak és telepítőkészletnek). A `Northwind`-adatbázis kiszolgálók jelentik a könyv legfőbb tesztkörnyezetét.

> **A `pubs` és `Northwind`-adatbázisokat külön kell telepíteni, amelyet a Microsoft webhelyére tölthetünk le. Az „F" függelékben bővebb információt találunk azzal kapcsolatban, hogyan telepítsük az adatbázisokat a gyakorló rendszerünkön.**

A tranzakciós napló

Hisszük vagy sem, a legtöbb dolog nem az adatbázisfájlban történik. Bár a rendszer az adatokat innen olvassa, az elvégzett változtatások nem az adatbázisba kerülnek. A rendszer folytatásosan a *tranzakciós naplóba* írja őket. Valamikor később az adatbázis kijelöl egy *ellenőrzőpontot,* ettől az időpillanattól a naplóban rögzített összes módosítás az aktuális adatbázisfájlba íródik.

Az adatbázis véletlen hozzáférésű elrendezéssel rendelkezik, de a napló soros természetű. Amíg az adatbázis véletlen elrendezése gyors hozzáférést biztosít, a napló soros volta lehetővé teszi a dolgok megfelelő sorrendben való nyomon követését. A napló összegyűjti azokat a módosításokat, amelyeket már jóváhagyottnak vél, majd egyszerre több módosítást a fizikai adatbázisfájl(ok)ba ír.

A módosítások naplózását részletesen a „Tranzakciók és zárak" című 14. fejezetben vizsgáljuk, most annyit jegyezzünk meg, hogy az adatok a lemezen először a naplóba kerülnek, majd az aktuális adatbázisba egy későbbi időpontban íródnak át. Működő adatbázishoz mind az adatbázisfájlra, mind pedig a tranzakciós naplóra szükség van.

A legalapvetőbb adatbázis-objektumok: tábla

Az adatbázisok több dolgot foglalnak magukban, de egyik összetevő sem játszik olyan központi szerepet az adatbázis felépítésében, mint a táblák. A táblára úgy gondolhatunk, mintha az egy könyvelő főkönyve vagy egy Excel-táblázat lenne. Úgynevezett *tartomány*adatokból (oszlopokból) áll, valamint *egyed*adatokból (sorokból). Az adatbázis tényleges adatai táblákban tárolódnak.

Minden egyes tábladefiníció metaadatokat is tartalmaz (leíró információk az adatokról), amelyek a tábla által tárol adatokról nyújtanak részleteket. Minden egyes oszlop saját szabálykészlettel rendelkezik a benne tárolható adatokra vonatkozóan. Bármely oszlop, bármely szabályának megszegésekor a rendszer visszautasíthatja egy sor beszúrását, vagy létező sor módosítását, illetve megakadályozhatja egy sor törlését.

Vizsgáljuk meg a `publishers` táblát a `pubs` adatbázisban. (A nézetet az SQL Server Management Studióból származó 1.1. ábrán látható nézet mutatja be. Ez egy alapvető eszköz, amelynek használatát a következő fejezetben mutatjuk be.

Az 1.1. ábrán látható tábla öt oszlopot tartalmaz. Az oszlopok száma attól függetlenül állandó, hogy a táblában mennyi adat van (a tábla akár üres is lehet). Pillanatnyilag a táblában nyolc rekord van. Az adatok hozzáadásával és törlésével a rekordok száma nő vagy csökken, de a rekordok (illetve most sorok) adatait leírja, és egyben korlátozza is az oszlop *adattípusa.*

pub_id	pub_name	city	state	country
0736	New Moon Books	Boston	MA	USA
0877	Binnet & Hardley	Washington	DC	USA
1389	Algodata Infosy...	Berkeley	CA	USA
1622	Five Lakes Publis...	Chicago	IL	USA
1756	Ramona Publishers	Dallas	TX	USA
9901	GGG&G	M□nchen	*NULL*	Germany
9952	Scootney Books	New York	NY	USA
9999	Lucerne Publishing	Paris	*NULL*	France

1.1. ábra

Szeretném megragadni az alkalmat, hogy kirohanást intézzek az objektumok névadásával kapcsolatban. Az SQL Server lehetővé teszi szóközök beágyazását a nevekbe, és néhány esetben kulcsszavakat is használhatunk objektumok neveként. Álljunk ellen a csábításnak, és ezt soha ne tegyük! Azok az objektumok, amelyek neve beágyazott szóközöket tartalmaz, SELECT utasításokban csinos fejlécet kapnak, de ugyanezt más eszközökkel is elérhetjük. A beágyazott szóközök és kulcsszavak használata az oszlopnevekben garantáltan hibákhoz, keveredéshez és más problémákhoz vezet. A későbbiekben majd visszatérek rá, hogy a Microsoft ezt miért hagyta jóvá, de most annyit jegyezzünk meg, hogy a nevekbe beágyazott szóközök vagy kulcsszavak használatát gonosz birodalmakkal, kínzással és a biztos halál képével kapcsoljuk össze. (Ez nem az utolsó alkalom, hogy ezt megjegyzem.)

Indexek

Az index olyan objektum, amely kizárólag egy adott táblához vagy nézethez kapcsolódik. Az index egy enciklopédia tárgymutatójához hasonlóan működik; vannak bizonyos szempontok szerint rendezett keresési (vagy „kulcs") értékek, ezáltal egy újabb kulcs áll rendelkezésünkre, amely segítségével a keresett információkat fellapozhatjuk.

Az index segítségével az információ fellapozását gyorsíthatjuk. Az indexek kétfélék lehetnek:

- **Csoportosított** – Táblánként csak egy ilyen indexet hozhatunk létre. A csoportosított index annyit jelent, hogy a tábla, amelyen a csoportosított index alapul, fizikailag az indexnek megfelelően rendezett. Ha egy enciklopédiát indexelnénk, a csoportosított index az oldalszám lenne; az enciklopédia az információkat oldalszám szerint tárolja.
- **Nem csoportosított** – Táblánként több ilyen indexünk is lehet. Valószínűleg ez jobban egybevág azzal az elképzeléssel, ami az „index" szó hallatán eszünkbe jut. Az ilyen index valamilyen más értékre mutat, amely segítségével megkereshetjük az adatokat. Az enciklopédiánk esetében ez lenne a könyv végén található kulcsszó index.

Jegyezzük meg, hogy az olyan nézetek esetében, amelyek indexekkel rendelkeznek – vagy *indexelt nézetek* –, először legalább egy csoportosított indexnek léteznie kell, mielőtt bármilyen nem csoportosított indexet létrehozhatnánk az adott nézeten.

Triggerek

A *trigger* olyan objektum, amely kizárólag a táblára definiálható. A triggerek logikai kódrészletek, amelyeket a rendszer automatikusan végrehajt, ha bizonyos események, például beszúrás, módosítás vagy törlés történik a táblában.

A triggerek több feladat esetében is hasznosak, de fő funkciójuk az adatmásolás, az adatok bevitele vagy frissítése során annak ellenőrzésére, hogy az adatok bizonyos feltételeknek megfelelnek-e.

Megszorítások

A *megszorítás* szintén olyan objektum, amely a táblát meghatározó keretek között létezik. A megszorítások nagyjából azok, amit a nevük sugall; korlátozzák a tábla adatai, hogy azok bizonyos feltételeknek megfeleljenek. A megszorítások a triggerek versenytársai abban a tekintetben, hogy lehetséges megoldást biztosítanak adatintegritási problémákra. Azonban a két dolog nem egyforma; mindegyik rendelkezik a maga előnyeivel és hátrányaival.

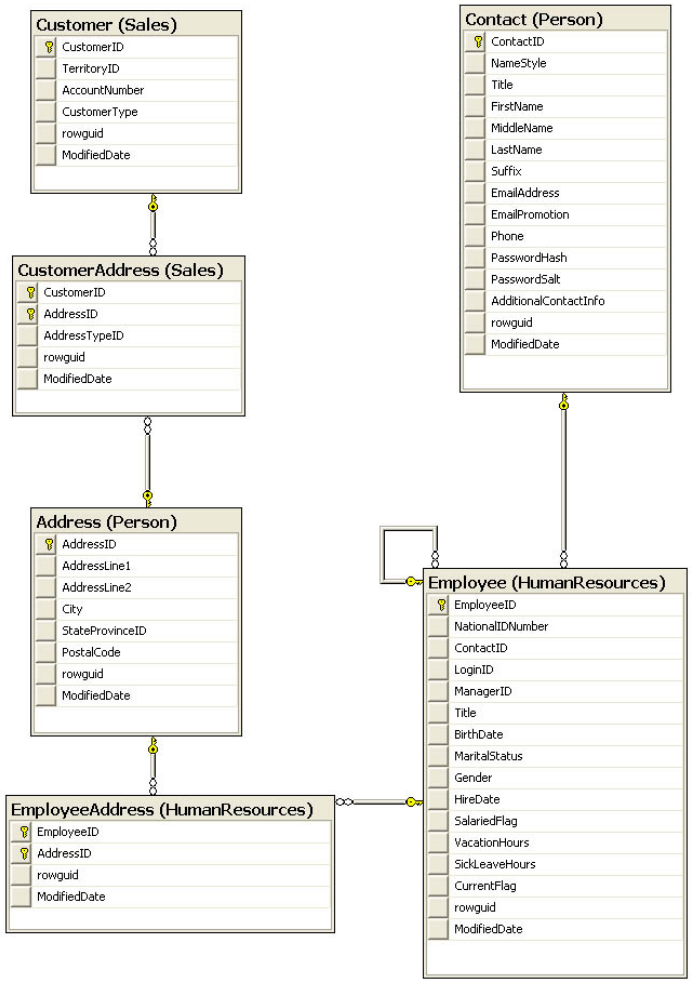
Fájlcsoportok

Alapértelmezés szerint minden táblát, valamint az adatbázis minden alkotóelemét (a naplót kivéve) egyetlen fájl tárol. A fájl az úgynevezett *elsődleges fájlcsoport* tagja. De nem kell ragaszkodnunk ehhez az elrendezéshez.

Az SQL Server lehetővé teszi valamivel kevesebb, mint 32 000 *másodlagos fájl* definiálását. (Ha többre van szükségünk, akkor az lehet, hogy nem az SQL Server problémája.) A másodlagos fájlokat az elsődleges fájlcsoporthoz adhatjuk, vagy létrehozhatjuk egy vagy több fájlcsoport másodlagos fájlcsoport részeként. Míg csupán egyetlen elsődleges fájlcsoportot hozhatunk létre (amelynek neve is „Primary”), addig 255 másodlagos fájlcsoporttal rendelkezhetünk. A másodlagos fájlcsoportot a `CREATE DATABASE` vagy `ALTER DATABASE` parancsok opcióinak segítségével hozhatjuk létre.

Diagrammok

Az adatbázisdiagram-rajzolással akkor foglalkozunk részleteiben, amikor a normalizálást és az adatbázis-tervezést tárgyaljuk, de most elegendő annyit tudnunk, hogy az adatbázis-diagram az adatbázis-tervezés képi megjelenítése, amely különböző táblákat, a táblákban oszlopneveket, és a táblák közötti kapcsolatokat foglalja magában. Fejlesztőként járva a világot talán hallottunk már az egyedkapcsolat diagramról, másnéven ERD-ről. Egy ERD-ben az adatbázis két részből áll: egyedekből (mint például a „szállító" és a „termék") és kapcsolatokból (mint például a „beszerzések" és az „eladások").



1.2. ábra

Annak ellenére, hogy az SQL Server 2005-ben az adatbázis-tervező eszközök teljes áttervezésen mentek keresztül, továbbra is kissé hiányosak. Az eszközök által használt diagramrajzolási módszer nem ragaszkodik az egyedkapcsolat diagramok elfogadott szabványainak egyikéhez sem.

Ennek ellenére ezek a diagramrajzoló eszközök minden „szükséges" dolgot biztosítanak; legalább kiindulási alapnak jók. Az ERD-vel és más eszközökkel kapcsolatban lásd a „C" függeléket.

Az 1.2. ábrán az AdventureWorks adatbázis néhány tábláját látjuk. A diagram (bár első pillantásra összetettnek tűnhet) az adatbázis több tulajdonságát is bemutatja. Vegyük észre a kulcsok kis ikonjait és a végtelenjelet. Ezek a két tábla közötti kapcsolatot jelenítik meg. A kapcsolatokról részletesebben a 7. és a 8. fejezetekben szólunk, és a könyv későbbi részében a diagramokra is visszatérünk.

Nézetek

A nézetek egyfajta virtuális táblák. A nézetet az esetek nagy részében a táblákhoz hasonlóan használjuk, de a nézet nem rendelkezik saját adatokkal. Ehelyett a nézet csupán a táblában tárolt adatok egy előre megtervezett vetítése és megjelenítése. A tervet lekérdezés formájában az adatbázis tárolja. A lekérdezés egy vagy több tábla néhány, nem feltétlenül az összes oszlopából kér adatokat. A visszakapott adatoknak esetenként (a nézet definíciójától függően) meg kell felelnie bizonyos követelményeknek, hogy azok a nézetben megjelenhessenek.

Az SQL Server 2000 megjelenéséig a nézetek elsődleges célja annak szabályozása volt, hogy a felhasználók milyen adatokat láthatnak. Ez két jelentős dolgot biztosít: biztonságot és a könnyű használatot. A nézetek segítségével szabályozhatjuk, hogy a felhasználók mit láthatnak, ha tehát a tábla egy része olyan adatokat tartalmaz, amelyhez csak néhány felhasználó férhet hozzá (például munkabér adatok), létrehozhatunk egy nézetet, amely csak azokat az oszlopokat tartalmazza, amelyekhez mindenki hozzáférhet. Ezen kívül a nézetet úgy testre szabhatjuk, hogy a felhasználónak nem kell felesleges információk között keresgélnie.

A nézet alapvető funkcióin kívül rendelkezésünkre áll annak lehetősége is, hogy *indexelt nézetet* hozzunk létre. Az ilyen nézetek mindenben szempontból megegyeznek a többi nézettel, azzal a kivétellel, hogy a nézet indexelhető. Ez a tény befolyásolja a teljesítményt (pozitívan, negatívan egyaránt):

- A több táblára hivatkozó nézetek általában sokkal jobb teljesítményt biztosítanak indexelt nézet segítségével, mert a táblák közötti összekapcsolást a rendszer előre elkészíti.

- A nézeten végrehajtott aggregációkat a rendszer előre kiszámítja és az index részeként tárolja; ez annyit jelent, hogy az aggregációt egyszer kell végrehajtani (sorok beszúrása vagy módosítása során), és a későbbiekben az indexinformációk segítségével közvetlenül olvasható.
- A beszúrások és törlések többletköltsége jóval nagyobb, mivel a nézet indexét is azonnal módosítani kell; a módosítások többletköltsége is számottevő, ha a módosítás érinti az index kulcsoszlopát is.

Ezekkel a teljesítményproblémákkal részletesebben a 10. fejezetben foglalkozunk.

Tárolt eljárások

A *tárolt eljárások* (stored procedures vagy más néven *sprocs)* történelmi múlttal rendelkeznek, és még a .NET korszakban is az SQL Server programozott működésének savát-borsát jelentik. A tárolt eljárások általában Transact-SQL (a Microsoft SQL Server lekérdezésére használt nyelv) utasítások rendezett sorozatából állnak, amelyek egyetlen logikai egységet alkotnak. Lehetővé teszik változók és paraméterek használatát, valamint elágazások és ciklusok építését. A kiszolgálóra egyesével elküldött utasítások végrehajtásával szemben a tárolt eljárások több előnyös tulajdonsággal rendelkeznek, mivel:

- Rövid névvel hivatkozhatunk rájuk, nem kell hosszú sztringeket használnunk, ezért az eljáráson belüli kód futtatása kisebb hálózati forgalmat generál.
- Optimalizált és előre fordított kódok, ezért az eljárások futtatása során minden egyes alkalommal megtakarítunk egy kis időt.
- Beskatulyáznak egy folyamatot, általában biztonsági okokból vagy csak azért, hogy az adatbázis összetettségét elrejtsék.
- Másik eljárásból is lehet őket hívni, tehát korlátozott mértékben újra felhasználhatók.

Ezen kívül bármely .NET-nyelv segítségével a T-SQL natív elemeken kívül újabb programelemeket adhatunk tárolt eljárásainkhoz.

Felhasználói függvények

A felhasználói függvények (user defined functions vagy UDF-ek) sok tekintetben hasonlóak a tárolt eljárásokhoz, de:

- Visszatérési értékük van, aminek típusa a legtöbb SQL Server adattípus lehet, kivéve a következő adattípusok: text, ntext, kép, kurzor és időbélyeg.
- Nincsenek „mellékhatásai". Alapvetően nem képesek semmire, ami a függvény hatókörén kívül esik, például táblák módosítására, e-mailek küldésére vagy rendszer, illetve adatbázis-paraméterek módosítására.

A felhasználói függvények hasonlítanak a standard programozási nyelvek, például a VB.NET vagy a C++ függvényeihez abban, hogy egynél több változót adhatunk meg bemenő paraméterként, és egy értéket kapunk vissza. Azonban az SQL Server felhasználói függvényei a legtöbb procedurális nyelv függvényeitől különböznek abban, hogy *minden,* a függvénynek átadott változó érték szerint kerül átadásra. Ha ismerjük a VB `By Ref` bemenő változóját, vagy a C++ mutatóinak átadását, akkor sajnálattal közlöm, hogy ennek itt nincs megfelelője. De jó hír, hogy visszaadhatunk egy speciális adattípust, a táblát. A 13. fejezetben ezt részletesen is megvizsgáljuk.

Felhasználók és szerepek

Ez a két objektum kéz a kézben jár. A *felhasználók* leginkább a bejelentkezések megfelelői. Röviden ez az objektum egy azonosítót jelent az SQL Serverbe bejelentkező személynek. Mindenki, aki bejelentkezik az SQL Serverbe (az érvényben levő adatvédelmi modelltől függően közvetlenül vagy közvetve) letérképezhető egy felhasználóra. A felhasználók pedig egy vagy több *szerephez* tartoznak. Az SQL Server bizonyos műveleteinek elvégzéséhez szükséges jogokat közvetlenül felhasználó vagy szerep számára engedélyezhetünk.

Szabályok

A szabályok és megszorítások a táblában tárolható adatokat szűrik. Ha egy módosított vagy beszúrt rekord megsért egy szabályt, az adatbázis visszautasítja a beszúrást vagy módosítást. A szabály *felhasználói adattípust* is korlátozhat. A szabályoktól eltérően a megszorítások nem igazi objektumok, inkább az adott táblát leíró metaadatok.

A szabályok csak a visszafelé kompatibilitás miatt állnak rendelkezésünkre, új fejlesztésekben kerüljük a használatukat.

Alapbeállítások

Kétféle alapbeállítás létezik. Az egyik alapbeállítás magában is objektum, a másik pedig nem, sokkal inkább a tábla adott oszlopát leíró metaadat (a szabályokhoz hasonlóan, amelyek objektumok, és a megszorításokhoz, amelyek nem objektumok, hanem metaadatok). Mindkét típus ugyanazt a célt szolgálja. Ha rekord beszúrása során az egyik oszlop értékét nem határozzuk meg, és az oszlop alapbeállítással rendelkezik, automatikusan az alapbeállításban meghatározott érték kerül az oszlopba. A két alapbeállítást a 6. fejezetben vizsgáljuk meg részletesebben.

Felhasználói adattípusok

A felhasználói adattípusok a rendszeradattípusok kibővítései. Az SQL Server 2005 verziójától kezdődően a lehetőségek csaknem végtelenek. Bár az SQL Server 2000 és a korábbi verziók már foglalkoztak a felhasználói adattípusok ötletével, de azok alapvetően a már létező adattípusok különböző szűrését jelentették. Az SQL Server 2005 esetében .NET-gyűjteményeket köthetünk saját adattípusainkhoz, azaz olyan adattípussal rendelkezhetünk, amely (értelmes kereteken belül) bármilyen .NET-objektumban is tárolható objektumot tárol.

> *Ezzel legyünk nagyon óvatosak! Az adattípusok, amelyekkel dolgozunk, az adataink és azok tárolásának szempontjából alapvető fontosságúak. Remek dolog, hogy saját adattípust definiálhatunk, de ne feledjük, hogy ennek hatalmas teljesítményköltsége is van. Alaposan gondoljuk át a dolgot, győződjünk meg róla, hogy valóban szükségünk van rá, és mint minden ilyen kísérletkor TESZTELNI, TESZTELNI, TESZTELNI!!!*

Teljes szövegű katalógusok

A teljes szövegű katalógusok az oszlopokon belül a szövegblokkok keresését gyorsító adatleképezések, és olyan oszlopokon működnek, amelyeken a teljes szövegű keresés engedélyezett. Ezek az objektumok a letérképezett táblákhoz és oszlopokhoz szorosan kötődnek, mégis különálló objektumok, és a rendszer nem módosítja őket automatikusan, ha az adatbázisban valamilyen változás történik.

SQL Server adattípusok

Most, hogy megismerkedtünk az SQL Server adatbázis alapszintű objektumaival, nézzük meg, hogy az SQL Server milyen lehetőségeket kínál az adattípusok kezelésére, amely bármely adatkezelő környezet alapvető eleme. Mivel a könyv fejlesztők számára készült, és egyetlen fejlesztő sem élhet 1 percnél tovább az adattípusok megértése nélkül, feltételezem, hogy már tudjuk, az adattípusok hogyan működnek, és csak az SQL Server adattípusok részleteire vagyunk kíváncsiak.

Az SQL Server 2005 a következő táblázatban felsorolt belső adattípusokkal rendelkezik:

Adattípus neve	Osztály	Méret (bájtokban)	Az adatok tulajdonságai
`Bit`	Egész szám	1	A méret kissé félrevezető. A táblában az első `bit` adattípus egy bájtot foglal; a következő hét adattípus ugyanazt a bájtot használja. A null értékek engedélyezése még egy bájtot foglal.
`Bigint`	Egész szám	8	Azzal a ténnyel foglalkozik, hogy egyre gyakrabban egyre nagyobb és nagyobb számokat használunk. Lehetővé teszi a –263 és 263–1 közötti egész számok használatát. Ez plusz vagy mínusz 92 kvintillió közötti érték.
`Int`	Egész szám	4	A –2147483648 és 2147483647 közötti egész számok.
`SmallInt`	Egész szám	2	A –32 768 és 32 767 közötti egész számok.
`TinyInt`	Egész szám	1	A 0 és 255 közötti egész számok
`Decimal` vagy `Numeric`	Decimális/ Numerikus	Változó	Rögzített pontosság és felbontás –1038 –1 és 10 38–1 között. A két név azonos jelentésű.

Adattípus neve	Osztály	Méret (bájtokban)	Az adatok tulajdonságai
`Money`	Pénz	8	Pénzegységek –263 és 263 között, valamint a négy tizedes helynyi pontosság. Bármilyen pénzegység lehet, nem csak dollár.
`SmallMoney`	Pénz	4	Pénzegységek –214748,3648 és +214748,3647 között.
`Float` (az ANSI `Real` szinonimája)	Approximate Numerikus	Változó	A méretet és pontosságot meghatározó argumentummal rendelkezhet (például `Float(20)`). Az argumentum bitben, nem bájtban értendő. –1.79 E + 308 és 1.79 E + 308 között értékek.
`DateTime`	Dátum/Idő	8	Az 1753. január 1. és 9999. december 31. közötti dátum- és időadatok, három századmásodpercnyi pontossággal.
`SmallDate-Time`	Dátum/Idő	4	Az 1900. január 1. és 2079. június 6. közötti dátum- és időadatok egyperces pontossággal.
`Cursor`	Speciális numerikus	1	Kurzor mutatója. Bár a mutató 1 bájtot foglal, ne feledjük, hogy az eredményhalmaz, amelyből a tényleges kurzor felépül, szintén foglal memóriát. Hogy pontosan mennyit, az az eredményhalmaztól függ.
`Timestamp/ rowversion`	Speciális numerikus (bináris)	8	Speciális érték, amely az adott adatbázison belül egyedi. Az értékét az adatbázis automatikusan állítja a rekordok beszúrása vagy módosítása során akkor is, ha az `UPDATE` utasítás nem hivatkozott időbélyeg oszlopra. (Az időbélyegmező közvetlen módosítása nem engedélyezett.)

Adattípus neve	Osztály	Méret (bájtokban)	Az adatok tulajdonságai
`Unique-Identifier`	Speciális numerikus (bináris)	16	Speciális globálisan egyedi azonosító (GUID) Időben és térben egyedi azonosító.
`Char`	Karakteres	Változó	Rögzített hosszúságú karakteres adatok. A beállított értéknél rövidebb értékeket a rendszer szóközökkel a beállított hosszúságra bővíti. Nem Unicode-os adatok. A maximális hosszúság 8000 karakter.
`VarChar`	Karakteres	Változó	Változó hosszúságú karakteres adatok. Az értékeket a rendszer nem bővíti szóközökkel. Nem Unicode-os adatok. A maximális hosszúság 8000 karakter, de a `max` kulcsszóval jelezhetjük, hogy alapvetően nagyon nagy karakteres mezőről (2^31 bájtnyi adat) van szó.
`Text`	Karakteres	Változó	Visszamenőleges (legacy) támogatás az SQL Server 2005-től kezdve. Helyette használjuk a `varchar(max)` típust.
`NChar`	Unicode	Változó	Rögzített hosszúságú Unicode karakteres adatok. A beállított hosszúságnál rövidebb értékeket a rendszer szóközökkel bővíti. Maximális hosszúság 4000 karakter.
`NVarChar`	Unicode	Változó	Változó hosszúságú Unicode karakteres adatok. Az értékeket a rendszer nem bővíti. A maximális hosszúság 4000 karakter, de a `max` kulcsszóval jelezhetjük, hogy alapvetően nagyon nagy karakteres mezőről (2^31 bájtnyi adat) van szó.

Adattípus neve	Osztály	Méret (bájtokban)	Az adatok tulajdonságai
`Ntext`	Unicode	Változó	A Text adattípushoz hasonlóan csak visszamenőleges (legacy) támogatás miatt. Helyette használjuk az `nvarchar(max)` típust. Változó hosszúságú Unicode karakteres adatok
`Binary`	Bináris	Változó	Maximálisan 8000 bájtnyi rögzített hosszúságú bináris adat.
`VarBinary`	Bináris	Változó	Maximálisan 8000 bájtnyi változó hosszúságú bináris adat, de a `max` kulcsszóval jelezhetjük, hogy alapvetően nagyon nagy LOB mezőről (2^31 bájtnyi adat) van szó.
`Image`	Bináris	Változó	Kizárólag visszamenőleges (legacy) támogatás az SQL Server 2005-től kezdve. Helyette használjuk a `varbinary(max)` típust.
`Table`	Egyéb	Speciális	Az eredményhalmazokkal való munka során használatos, általában felhasználói függvények kimenő paramétere. Tábladefiníción belül nem használható adattípusként.
`Sql_variant`	Egyéb	Speciális	A VB és C++ `Variant` típusához hasonlít. Alapjában véve egy tároló, amely lehetővé teszi, hogy más SQL Server adattípusokat tároljunk benne. Ez azt jelenti, hogy akkor lesz rá szükségünk, ha egy oszlop vagy függvény több adattípus kezelésére kell, hogy alkalmas legyen. A VB-től eltérően az adattípus használata rákényszerít bennünket az *explicit* meghatározásra, hogy a rendszer meghatározott adattípusra tudja konvertálni.

Adattípus neve	Osztály	Méret (bájtokban)	Az adatok tulajdonságai
XML	Karakteres	Változó	XML-adatokat magában foglaló karakteres mezőt határoz meg. Biztosítja az adatellenőrzést az XML-sémában, valamint a speciális XML-központú függvények használatát.

Az adattípusok nagy része más programozási nyelv adattípusaival ekvivalens. Például az SQL Server `int` típusa a Visual Basic `long` típusával ekvivalens, valamint a legtöbb rendszerben és C++ fordító kombinációban az `int` adattípussal ekvivalens.

Az SQL Server nem ismeri az előjel nélküli numerikus adatok fogalmát.

Általában az SQL Server adattípusok más modern programozási nyelvben szerzett tapasztalatainknak megfelelően működnek. A számok összeadásának eredménye egy összeg, de sztringek összeadása során az eredmény az összefűzött elem. Amikor különböző adattípusok változóinak vagy mezőinek használatát vagy értékadását összekeverjük, több adattípust a rendszer implicit módon (vagy automatikusan) konvertál. A többi adattípus nagy része explicit módon konvertálható. (Meghatározhatjuk, hogy milyen adattípusra szeretnénk konvertálni.) Van néhány adattípus, ami egyáltalán nem konvertálható. Az 1.3. ábra a különböző konvertálási lehetőségeket mutatja.

Miért van szükség az adatok konvertálására? Nézzünk egy egyszerű példát. Ha szeretném kiírni a következőket: „Today's date is ##/##/####", ahol ##/##/#### az aktuális dátum, az alábbi lekérdezésre lenne szükségem:

```
SELECT 'Today''s date is ' + GETDATE()
```

Az ehhez hasonló Transact-SQL utasításokra a könyv későbbi részében sokkal részletesebben visszatérünk, de az előző példa várt eredményének eléggé nyilvánvalónak kell lennie.

Az a probléma, hogy az utasítás a következő eredményt hozza:

```
Msg 241, Level 16, State 1, Line 1
Syntax error converting datetime from character string.
```

Ez nem igazán az, amire vágytunk, nem? Most próbálkozzunk a `CONVERT()` függvénnyel:

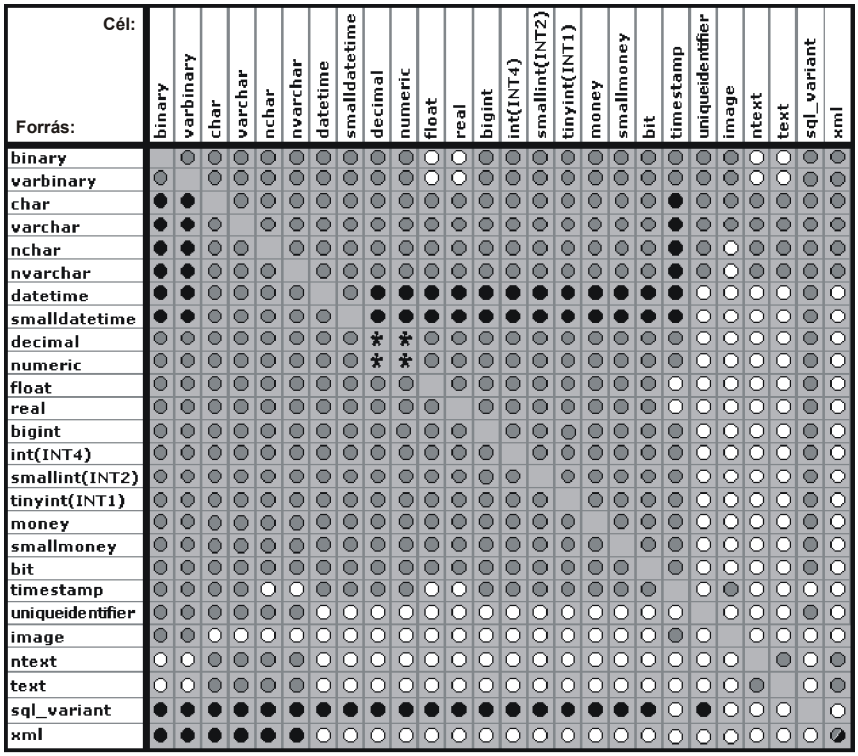
```
SELECT "Today's date is " + CONVERT(varchar(12), GETDATE(),101)
```

Ez a következő eredményt adja vissza:

```
---------------------------------
Today's date is 01/01/2000

(1 row(s) affected)
```

A dátum és idő adattípusok, például az GETDATE függvény kimenete például a `"Today's date is"` esetén, implicit módon nem konvertálható sztring adattípusra, mégis rendszeresen ilyen konverziókba ütközünk. Szerencsére a `CAST` és `CONVERT` függvények segítségével sok SQL Server adattípust konvertálhatunk. A `CAST` és `CONVERT` függvényekre egy későbbi fejezetben visszatérünk.



- ● Explicit konverzió
- ● Implicit konverzió
- ○ Nem megengedett konverzió
- * A CAST függvény explicit alkalmazása szükséges annak érdekében, hogy az implicit konverzió során a pontosság és a felbontás ne vesszen el.
- ◪ Az XML adattípusok közötti konverzió csak akkor támogatott, ha a forrás- vagy a céladatok típus nélküli xml-adatok. Különben a konverzió explicit.

1.3. ábra

Röviden összefoglalva az SQL Server adattípusai ugyanazokat a funkciókat töltik be, mint más programozási környezetek adattípusai. Segítik a programozási hibák elkerülését, mivel biztosítják, hogy a szolgáltatott adatok a vártnak megfelelőek (ne feledjük, hogy az 1/1/1980 dátumként és számként mást-mást jelent), és a rendszer az elvárt műveletet hajtja végre.

NULL adat

Mi van akkor, ha az egyik sorban egy adott oszlopnak nincsen értéke, azaz nem ismerjük az oszlop értékét? Vegyük például azt a rekordot, amely a vállalat teljesítményinformációit tárolja egy adott évre vonatkozóan. Képzeljük azt, hogy a mezők egyike a korábbi évhez képest a százalékos növekedést tárolja, de az előző évre vonatkozóan nincsenek adataink, mivel ez az adatbázis első rekordja. Késztetést érezhetünk arra, hogy a `PercentGrowth` oszlopba beírjunk egy nullát. De ezzel a helyes információt adnánk meg? Azok, akik nem gondolják végig a dolgot, azt hihetik, hogy nulla százalékos növekedést értünk el, amikor pusztán az a helyzet, hogy nem tudjuk az adott évben ezt az értékét.

A meghatározhatatlan értékeket `NULL` értékként kell kezelni. Úgy tűnik, hogy ahányszor programozási tanfolyamot tartok, legalább egy hallgató mindig megkér, hogy definiáljam a `NULL` értékét. Ez egy rázós kérdés, mivel definíció szerint a `NULL` érték azt jelenti, hogy nem tudjuk, mi az oszlop értéke. Lehet, hogy 1; lehet, hogy 347; de ismereteink szerint akár –294 is lehet. Röviden, *meghatározatlan* vagy *nem alkalmazható* értéket jelent.

Az objektumok SQL Server azonosítói

Mostanra már rengeteg dolgot hallottunk az SQL Server objektumairól. Az oszlopnevek kapcsán már az első hordószónoklatomon is túl vagyunk. De nézzük meg közelebbről az SQL Server objektumainak elnevezéseit.

Mi kap nevet?

Alapjában véve az SQL Serverben mindennek van neve. Íme egy – egyáltalán nem teljes – lista:

Tárolt eljárások	Táblák	Oszlopok
Nézetek	Szabályok	Megszorítások
Alapbeállítások	Indexek	Fájlcsoportok
Triggerek	Adatbázisok	Kiszolgálók
Felhasználói függvények	Bejelentkezések	Szerepek
Teljesszövegű katalógusok	Fájlok	Felhasználói adattípusok

És a lista még hosszan folytatódik. A sorok (amelyek nem igazán objektumok) kivételével mindennek, ami eszembe jut, van neve. A trükk lényege, hogy minden név hasznos és praktikus legyen.

Névadási szabályok

Ahogyan már a fejezet korábbi részében említettem, az SQL Server névadási szabályai eléggé lazák; megengedik a szóközök és a kulcsszavak használatát az objektumok elnevezésekor. A nagy szabadság eredményeként azonban könnyen hozunk rossz döntéseket, és saját magunkat keverjük bajba.

Íme a fő szabályok:

- Az objektum nevének a Unicode 2.0 specifikáció értelmében betűvel kell kezdődnie. Ez a nyugaton használt betűket jelenti A–Z és a–z között. Az, hogy az „A” és az „a” különbözik, a kiszolgáló beállításaitól függ, de mindkettő használható az objektumnév első betűjeként. Az első betű után már elszabadulhat a fantáziánk; szinte bármilyen karakter használható.
- Normál objektumok esetében a név 128 karakter hosszú lehet, átmeneti objektumok esetében 116 karakter.
- Bármely nevet, amely megegyezik valamely SQL Server kulcsszóval, vagy beágyazott szóközöket tartalmaz, dupla idézőjelek („”) vagy szögletes zárójelek ([]) közé kell tennünk. Az, hogy mely szavak számítanak kulcsszónak attól függ, hogy az adatbázis kompatibilitási szintjét mire állítottuk.

> *Ne feledjük, hogy a dupla idézőjelek csak akkor számítanak elválasztó jelnek, ha a* `SET QUOTED_IDENTIFIER ON` *beállítás van érvényben. A szögletes zárójelek (*`[` *és* `]`*) használatával elkerülhető, hogy a felhasználó rossz beállítás mellett dolgozzon.*

Ezek a szabályok az úgynevezett azonosító szabályok, amelyek az SQL Server objektumainak elnevezése során mindig érvényben vannak. Meghatározott objektumtípusok esetén még további szabályok is lehetnek.

Nem tudom eléggé hangsúlyozni annak fontosságát, hogy kerüljük az SQL Server kulcsszavak vagy beágyazott szóközök használatát az objektumnevekben. Bár technikailag mindkettő érvényes megoldás, ez a gyakorlat végtelen problémák forrása lehet.

Összegzés

Az élet legtöbb területéhez hasonlóan a kis dolgok is számítanak, amikor az RDBMS-ről beszélünk. Bizonyára majdnem mindenkinek, aki elegendő ismerettel rendelkezik ahhoz, hogy kezébe vegye ezt a könyvet, van valamilyen elképzelése az adatok oszlopokban és sorokban való tárolásának fogalmáról még akkor is, ha nem tudjuk, hogy az oszlopok és sorok csoportosításait táblának hívják, de egy igazi adatbázis nem csupán két táblából áll. Azok a dolgok, amelyek a mai RDBMS-eket nagyszerűvé teszik, az extra dolgok: az objektumok, amelyek lehetővé teszik, hogy az adatokhoz tartozó működést és üzleti szabályokat az adatokkal együtt az adatbázisban helyezzük el.

Az adatbázis adatai a legtöbb programozási környezethez hasonlóan típusokkal rendelkeznek. Az SQL Serverben végzett legtöbb műveletnek köze van a típusokhoz. Tekintsük át a rendelkezésünkre álló típusokat, és gondoljuk végig, hogy ezek a típusok hogyan felelnek meg az általunk ismert programozási környezetek adattípusainak.