SQL dla SQL Server 2005. Wprowadzenie

Autorzy: Sikha Saha Bagui, Richard Walsh Earp
Tłumaczenie: Piotr Pilch, Tomasz Nowak
ISBN: 83-246-0614-9
Tytuł oryginału: Learning SQL on SQL Server 2005
Format: B5, stron: 328

Rozpocznij przygodę z bazą danych SQL Server 2005

- Naucz się korzystać z SQL Server 2005
- Poznaj język SQL

We współczesnym przemyśle informatycznym coraz większą rolę odgrywają aplikacje bazodanowe, a SQL to podstawowy język służący do manipulowania bazami danych. Za jego pomocą można wykonać wszelkie potrzebne operacje, a poza tym jest on standardem w świecie przetwarzania danych. Bazują na nim niemal wszystkie najważniejsze systemy zarządzania bazami danych. Jednym z najbardziej zaawansowanych z nich jest Microsoft SQL Server 2005, który zapewnia niezawodną i wydajną obsługę aplikacji bazodanowych.

Książka „SQL dla SQL Server 2005. Wprowadzenie" stanowi doskonały wstęp do języka SQL i bazy danych Microsoft SQL Server 2005. Dzięki niej dowiesz się, jak szybko rozpocząć pracę z SQL Server 2005. Poznasz podstawowe instrukcje i funkcje języka SQL oraz nauczysz się tworzyć, wypełniać, modyfikować, złączać i usuwać tabele, agregować dane i tworzyć zapytania. Przeczytasz także o sposobach dodawania do tabel przydatnych indeksów i ograniczeń. Każdy rozdział zakończony jest pytaniami i ćwiczeniami, co pozwoli Ci utrwalić zdobytą wiedzę zarówno w teorii, jak i w praktyce.

- Instrukcje języka SQL
- Korzystanie z SQL Server 2005
- Funkcje języka SQL
- Obsługa tabel
- Tworzenie zapytań
- Złączanie danych
- Agregowanie danych

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 3.

Tworzenie, wypełnianie, modyfikowanie i usuwanie tabel

W bazie danych serwera SQL Server 2005 dane są przechowywane w tabelach (w teorii relacyjnych baz danych określa się je też mianem **relacji**). W rozdziale 2. wyjaśniliśmy, jak tworzyć zapytania pobierające przy użyciu instrukcji `SELECT` dane z **istniejących** tabel. W tym rozdziale pokażemy, w jaki sposób tworzyć tabele i umieszczać w nich dane, a także jak za pomocą języka SQL modyfikować, aktualizować i usuwać tabele oraz ich dane. Na początku rozdziału omówimy typy danych. Zanim za pomocą instrukcji `CREATE TABLE` utworzy się tabele, trzeba się zapoznać z różnymi typami danych. Poza nazwami kolumn w instrukcji `CREATE TABLE` muszą być uwzględnione typy danych i rozmiary kolumn.

Typy danych serwera SQL Server 2005

Każda kolumna tabeli posiada typ danych, który określa, jakiego rodzaju informacje lub wartości mogą być przechowywane w kolumnie, a także jakie operacje mogą być wykonywane na tych danych. Jest to kwestia odwzorowywania wartości domenowych, które trzeba przechowywać, na odpowiadający im typ danych. Wybierając typ danych, powinno się unikać marnotrawienia przestrzeni dyskowej, a jednocześnie zadbać o to, aby była dostępna wystarczająca ilość miejsca dla odpowiedniego zakresu możliwych wartości, które mogą być użyte w całym okresie używania aplikacji. Zaprezentujemy najczęściej stosowane typy danych, dzieląc je na cztery kategorie — numeryczne, znakowe, czasu i daty oraz różne inne.

Wartości domenowe są zestawem wszystkich możliwych wartości, które mogą zostać zapisane w kolumnie. Na przykład w przypadku kolumny `GPA` wartości domenowe mogą zawierać się w przedziale od 0 do 4.

Kilka podstawowych typów danych posiada również obowiązujące synonimy, które mogą być użyte zamiast zwykłych typów danych. Synonimy są zewnętrznymi nazwami, których zadaniem jest sprawienie, aby jeden produkt SQL-owy było zgodny z innym.

Im bardziej zadba się o dokładność podczas wybierania dla kolumny typu danych, tym precyzyjniejsze będą informacje znajdujące się w bazie danych. W poniższych punktach w skrócie opisano każdy typ danych i jego poprawne synonimy.

Numeryczne typy danych

Numeryczne typy danych powinny być używane do przechowywania danych liczbowych, w przypadku których zamierza się wykonać operacje arytmetyczne lub porównywania wartości liczbowych. Numeryczne typy danych można podzielić na dwie grupy — całkowitoliczbowe i dziesiętne.

Całkowitoliczbowe typy danych

W przypadku całkowitoliczbowych typów danych po przecinku dziesiętnym nie występują żadne cyfry. Ponadto rozmiar wewnętrznego obszaru przechowywania dla tego typu danych zawiera się w przedziale od 1 do 8 bajtów. Można wyróżnić następujące całkowitoliczbowe typy danych serwera SQL Server 2005:

- `BIGINT`. Używa 8-bajtowego obszaru przechowywania i może być wykorzystany do magazynowania liczb z przedziału od -2^{63} do $2^{63}-1$. Jeśli naprawdę nie są potrzebne dodatkowe możliwości tego typu danych, dotyczące przechowywania, należy go unikać.
- `INT`. Stosuje 4-bajtowy obszar przechowywania i może posłużyć do magazynowania liczb z przedziału od -2^{31} do $2^{31}-1$.
- `SMALLINT`. Używa 2-bajtowego obszaru przechowywania i może być wykorzystany do magazynowania liczb z przedziału od -2^{15} do $2^{15}-1$.
- `TINYINT`. Stosuje 1-bajtowy obszar przechowywania i może posłużyć do magazynowania liczb z przedziału od 0 do 255.
- `MONEY`. Dysponuje 8-bajtowym obszarem przechowywania.
- `SMALLMONEY`. Stosuje 4-bajtowy obszar przechowywania.

Typy danych `MONEY` i `SMALLMONEY` zostały dodane do całkowitoliczbowych typów danych, ponieważ wewnętrznie są przechowywane w taki sam sposób jak inne typy całkowitoliczbowe.

`INTEGER` jest synonimem dla typu danych `INT`.

Dziesiętne typy danych

Dziesiętne typy danych oferują szerszy zakres wartości i większą dokładność niż całkowitoliczbowe typy danych. W przypadku dziesiętnych typów danych można określić precyzję i skalę. **Precyzja** jest całkowitą liczbą przechowywanych cyfr, natomiast **skala** maksymalną liczbą cyfr znajdujących się z prawej strony przecinka dziesiętnego. Obszar przechowywania dziesiętnego typu danych jest zależny od precyzji. Liczby dziesiętne o precyzji z przedziału od 1 do 9 wymagałyby maksymalnie 5-bajtowego obszaru. Z kolei wartości dziesiętne o precyzji od 10 do 19 potrzebowałyby maksymalnie 9-bajtowego obszaru itd.

Istnieją następujące dziesiętne typy danych:

- `REAL`. Stosuje 4-bajtowy obszar przechowywania i posiada 7-cyfrową precyzję. Synonimem typu `REAL` jest `FLOAT[(n)]`, gdzie *n* przyjmuje wartości od 1 do 7.
- `FLOAT`. Używa 8-bajtowego obszaru przechowywania i posiada 15-cyfrową precyzję. Synonimem typu `FLOAT` jest `DOUBLE PRECISION` i `FLOAT[(n)]`, gdzie *n* przyjmuje wartości od 8 do 15.
- `DECIMAL`. Rozmiar obszaru przechowywania zmienia się zależnie od podanej precyzji i zawiera się w przedziale od 2 do 17 bajtów. Synonimem typu `DECIMAL` jest `DEC` i `NUMERIC`.

Błędy zaokrąglania mogą wystąpić, gdy zastosuje się typ danych `FLOAT` lub `REAL`. W takich sytuacjach lepszy jest typ danych `NUMERIC` lub `DECIMAL`, ponieważ oferuje precyzję i skalę bez występowania problemów związanych z typem danych `FLOAT` lub `REAL`.

Gdy próbuje się wybrać numeryczny typ danych, decyzja powinna być zależna od maksymalnego zakresu możliwych wartości, które zamierza się przechowywać, a także żądanej precyzji i skali. Jednak jednocześnie trzeba mieć świadomość, że typy danych, które są w stanie magazynować większy zakres wartości, wymagają większej przestrzeni dyskowej.

Typ danych `NUMERIC` najbardziej przypomina typ danych `NUMBER` serwera bazodanowego firmy Oracle.

Znakowe typy danych

Znakowe typy danych są wykorzystywane do przechowywania dowolnej kombinacji liter, liczb i symboli. Przy wprowadzaniu danych znakowych trzeba zastosować apostrofy. Serwer SQL Server 2005 posiada pięć rodzajów znakowych typów danych — `CHAR`, `VARCHAR`, `TEXT`, `NCHAR` i `NVARCHAR`.

Typ danych CHAR

Zapis `CHAR(n)` identyfikuje 1-bajtowe łańcuchy znakowe o stałej długości, które mogą posłużyć do przechowania maksymalnie 8000 bajtów. Typ danych `CHAR` jest stosowany, gdy długość kolumny jest znana i niezmienna. Na przykład numer ubezpieczenia społecznego może wykorzystywać typ danych `CHAR(9)`. Ponieważ typ danych `CHAR` używa obszaru przechowywania o stałym rozmiarze, w porównaniu z typem `VARCHAR` (łańcuchy znakowe o zmiennej długości) oferuje szybszy dostęp. Można i powinno się za pomocą wartości *n* określić dla typu danych `CHAR(n)` maksymalną długość bajtową. W przeciwnym razie zostanie zastosowany domyślny rozmiar, który może być znacznie większy od żądanego. `CHARACTER` jest synonimem typu danych `CHAR`.

Typ danych VARCHAR

Zapis `VARCHAR(n)` identyfikuje 1-bajtowe łańcuchy znakowe o zmiennej długości, które mogą być wykorzystane do przechowania maksymalnie 8000 bajtów. Można i powinno się za pomocą wartości *n* określić dla typu danych `VARCHAR(n)` maksymalną długość bajtową. W przeciwnym razie, podobnie jak w przypadku typu danych `CHAR`, zostanie zastosowany domyślny

rozmiar, który może być znacznie większy od wymaganego. Zmienna długość oznacza, że jeśli zastosuje się dane o rozmiarze mniejszym od określonych *n* bajtów, wielkość obszaru przechowywania będzie równa rzeczywistej długości wprowadzonych danych. CHAR VARYING jest synonimem typu danych VARCHAR. VARCHAR jest najczęściej używanym znakowym (łańcuchowym) typem danych.

Typ danych VARCHAR2 obecny w serwerze bazodanowym firmy Oracle jest odpowiednikiem typu VARCHAR.

Typ danych TEXT

Zapis TEXT również identyfikuje 1-bajtowe łańcuchy znakowe o zmiennej długości, które mogą być wykorzystane do przechowania więcej niż 8000 bajtów. W przypadku serwera SQL Server 2005 typ danych TEXT jest powiązany z dużymi obiektami i lepiej nadaje się do magazynowania dużych łańcuchów danych. Typ TEXT powoduje dodatkowe obciążenie, które zmniejsza wydajność. W związku z tym użycie tego typu danych nie jest wskazane.

Typ danych LONG serwera bazodanowego firmy Oracle jest odpowiednikiem typu TEXT.

Typ danych NCHAR

Zapis NCHAR reprezentuje łańcuchy znakowe Unicode o stałej długości. Za pomocą wartości *n* w przypadku tego typu danych można też określić maksymalną długość bajtową. NATIONAL CHAR jest synonimem typu NCHAR.

Typ danych NVARCHAR

Zapis NVARCHAR identyfikuje łańcuchy znakowe Unicode o zmiennej długości. Za pomocą wartości *n* w przypadku tego typu danych można określić maksymalną długość bajtową. NATIONAL CHARACTER VARYING jest synonimem typu NVARCHAR.

Łańcuchy znakowe Unicode

Dla każdego przechowywanego znaku łańcuchy znakowe Unicode wymagają dwóch bajtów. Jednak większość anglojęzycznych i innych europejskich alfabetów może być zapisana za pomocą 1-bajtowych znaków. Łańcuchy złożone z 1-bajtowych znaków mogą przechowywać maksymalnie 8000 znaków, natomiast łańcuchy znakowe Unicode 4000 znaków.

Wybieranie znakowych typów danych

W celu stwierdzenia, jaki zastosować znakowy typ danych, można kierować się następującymi kilkoma ogólnymi regułami:

- Gdy oczekuje się mnóstwa wartości pustych lub znacznego zróżnicowania danych pod względem wielkości, zamiast typu danych o stałej długości CHAR należy zastosować typ o zmiennej długości VARCHAR.
- Jeśli liczba znaków zawartych w kolumnie nie ulega znacznym wahaniom, pod uwagę należy wziąć użycie typu danych CHAR zamiast typu VARCHAR.
- Jeżeli nie trzeba przechowywać 16-bitowych danych znakowych Unicode, typ danych NVARCHAR lub NCHAR nie powinien być wykorzystywany. Oba typy danych zajmują dwukrotnie więcej miejsca niż typy VARCHAR i CHAR, powodując spadek wydajności operacji wejścia-wyjścia.

Typy danych czasu i daty

Serwer SQL Server 2005 oferuje dwa typy danych służące do przechowywania czasu i daty — DATETIME i SMALLDATETIME. Typ danych DATETIME używa 8 bajtów, natomiast typ SMALLDATETIME 4 bajty obszaru przechowywania. Wewnętrznie wartości obu typów danych są magazynowane w zupełnie innej postaci niż stosowana podczas ich wprowadzania lub wyświetlania. Wartości są zapisywane jako dwa oddzielne składniki — daty i czasu.

Typ danych DATE obecny w serwerze bazodanowym firmy Oracle jest odpowiednikiem typu DATETIME.

Tworząc klucze podstawowe, nie należy brać pod uwagę zastosowania typów danych DATETIME i SMALLDATETIME. Z punktu widzenia wydajności lepiej użyć typu danych, który na potrzeby klucza podstawowego zużywa mniej przestrzeni dyskowej. Im mniej miejsca przeznaczy się na ten cel, tym mniejsza będzie tabela i indeks, a także obciążenie związane z operacjami wejścia-wyjścia niezbędnymi do uzyskania dostępu do klucza podstawowego.

Definiowanie kluczy podstawowych zostanie przedstawione w rozdziale 11.

Inne typy danych

Wśród innych typów danych dostępnych w serwerze SQL Server 2005 należy wymienić BINARY, IMAGE, BIT, TABLE, SQL_VARIANT, UNIQUEIDENTIFIER i XML (jedno z najnowszych rozszerzeń tej wersji serwera).

Typy danych BINARY

Można wyróżnić dwa typy danych BINARY — BINARY i VARBINARY.

Typy danych BINARY są wykorzystywane do przechowywania ciągów bitów i wartości, które są wprowadzane i wyświetlane za pomocą zapisu heksadecymalnego. Maksymalna wielkość typu danych BINARY to 8000 bajtów. W celu określenia maksymalnej długości bajtowej danych typu BINARY należy użyć wartości *n*.

Typ danych VARBINARY pozwala zapisać maksymalnie 8000 bajtów binarnych danych o zmiennej długości. Również w tym przypadku maksymalną długość bajtową można zdefiniować za pomocą wartości *n*. Typ danych VARBINARY powinien być użyty zamiast typu BINARY, gdy oczekuje się wystąpienia wartości pustych lub o zmiennym rozmiarze.

Typ danych RAW serwera bazodanowego firmy Oracle jest odpowiednikiem typu VARBINARY.

Typ danych IMAGE

Typ danych IMAGE jest powiązany z dużymi binarnymi obiektami, które zajmują więcej niż 8000 bajtów. Typ ten służy do przechowywania binarnych wartości, a także obrazów.

Typ danych LONG RAW serwera bazodanowego firmy Oracle jest odpowiednikiem typu IMAGE.

Typ danych BIT

Typ danych BIT jest w rzeczywistości całkowitoliczbowym typem danych, który może jedynie przechowywać wartość 0 lub 1, a ponadto wykorzystuje tylko 1 bit. Jeśli jednak w tabeli występuje 1-bitowa kolumna, faktycznie zajmie cały bajt. Kolumny o maksymalnym rozmiarze wynoszącym 8 bitów są przechowywane przy użyciu jednego bajta. Typ danych BIT jest zazwyczaj stosowany na potrzeby wartości prawda — fałsz lub tak — nie. Kolumny typu BIT nie mogą być NULL i nie można uwzględniać ich w indeksach.

Monetarne typy danych

Monetarne typy danych zwykle są używane do przechowywania wartości walutowych. Serwer SQL Server 2005 posiada następujące dwa monetarne typy danych:

- MONEY. Wykorzystuje 8-bajtowy obszar przechowywania.
- SMALLMONEY. Stosuje 4-bajtowy obszar magazynowania.

Typ danych TABLE

Typ danych TABLE może posłużyć do przechowywania wyniku funkcji i być zastosowany dla lokalnych zmiennych. Jednak kolumny tabel nie mogą być typu TABLE. Zmienne tabelowe czasami są preferowane w przypadku tymczasowych tabel, ponieważ takie zmienne są automatycznie czyszczone na końcu wykonywanej funkcji lub procedury przechowywanej.

Tabele tymczasowe przedstawiono w rozdziale 6. Omówienie funkcji i procedur przechowywanych wykracza poza zakres tej książki.

Typ danych SQL_VARIANT

Wartości znajdujące się w kolumnie typu `SQL_VARIANT` mogą być dowolnego typu danych z wyjątkiem `TEXT` lub `IMAGE`. Z użycia typu danych `SQL_VARIANT` należy zrezygnować z następujących kilku powodów: (a) kolumna typu `SQL_VARIANT` nie może wchodzić w skład klucza podstawowego lub obcego; (b) kolumna typu `SQL_VARIANT` nie może być częścią kolumny obliczeniowej; (c) kolumna typu `SQL_VARIANT` może być użyta w indeksach lub w roli innych unikatowych kluczy tylko wtedy, gdy nie przekroczą 900 bajtów; (d) kolumna typu `SQL_VARIANT` musi dokonać konwersji typu danych na inny, gdy przenosi się je do obiektów innego typu.

Klucze obce zaprezentowano w rozdziale 11.

Typ danych UNIQUEIDENTIFIER

Typ danych `UNIQUEIDENTIFIER` nazywany też globalnym unikatowym identyfikatorem GUID (*Globally Unique Identifier*) lub uniwersalnym unikatowym identyfikatorem UUID (*Universal Unique Identifier*) jest 128-bitową generowaną wartością, która gwarantuje niepowtarzalność w skali światowej, nawet wśród komputerów niepodłączonych do sieci.

Typ danych XML

`XML` jest nowym typem danych dodanym do serwera SQL Server 2005 w celu obsługi danych XML. Typ `XML` może modelować złożone dane. Kolumna typu `XML` może być ogólna lub dokładna. Podobnie do innych typów danych, typ `XML` musi spełniać określone kryteria związane z formatowaniem. Musi być zgodny z wymaganiami dotyczącymi poprawnie sformatowanego dokumentu XML (ogólność). Można ponadto opcjonalnie zdefiniować dodatkowe kryterium zgodności, określając zbiór schematu (dokładność). Serwer SQL Server umożliwia też przechowywanie dokumentów XML powiązanych z wieloma definicjami schematów. Typ danych `XML` pozwoli zapisać kompletne dokumenty XML lub ich fragmenty. Maksymalny rozmiar dokumentów XML wynosi 2 gigabajty.

Wybieranie typów danych

Poniżej przedstawiono kilka ogólnych zasad, które należy uwzględnić w celu stwierdzenia, jakiego typu danych użyć podczas definiowania kolumny. Oto one:

- Dla kolumn należy określać najmniejszy możliwy rozmiar. Im mniejsza wielkość kolumny, tym mniej danych serwer SQL Server będzie musiał przechowywać i przetwarzać. Ponadto serwer będzie w stanie szybciej wczytywać i zapisywać dane. Dodatkowo mniejsza kolumna sprawi, że wykonywana dla niej operacja sortowania będzie szybciej zrealizowana.
- Dla kolumny, która będzie przechowywać dane, należy użyć typu danych o najmniejszym możliwym zapotrzebowaniu na przestrzeń dyskową. Jeśli na przykład zamierza się umieścić w kolumnie wartości z przedziału od 1 do 99, zamiast typu danych `INT` lepsze będzie wybranie typu `TINYINT`.

- W przypadku danych numerycznych lepsze będzie zastosowanie numerycznego typu danych, takiego jak `INTEGER`, zamiast typu `VARCHAR` lub `CHAR`. Wynika to stąd, że w porównaniu ze znakowymi typami danych typy numeryczne zwykle wymagają mniej miejsca do przechowywania wartości liczbowych. W ten sposób oszczędza się przestrzeń dyskową. Mniejsze kolumny mogą przyczynić się do wzrostu wydajności, gdy są przeszukiwane, złączane z innymi kolumnami lub sortowane.

Złączenia omówiono w rozdziale 4.

- Typ danych `FLOAT` lub `REAL` nie powinien być stosowany do definiowania kluczy podstawowych. W tym celu można wykorzystać całkowitoliczbowe typy danych.
- Jeśli w kolumnie będzie wiele wartości pustych, należy unikać wybierania dla niej typu danych `CHAR` lub `NCHAR` o stałej długości. Wartość `NULL` umieszczona w polu kolumny typu danych `CHAR` lub `NCHAR` wykorzysta całą jego stałą długość wynoszącą 255 znaków. W ten sposób zmarnuje się wiele przestrzeni i spowoduje spadek ogólnej wydajności serwera SQL Server.
- Jeżeli kolumnę często będzie się wykorzystywać na potrzeby sortowania, zamiast znakowego typu danych warto zastanowić się nad użyciem dla niej całkowitoliczbowego typu danych. Serwer SQL Server sortuje dane całkowitoliczbowe szybciej niż znakowe[1].

Tworzenie tabeli

W przypadku serwera relacyjnych baz danych SQL Server 2005 dane są umieszczane w tabelach tworzonych w bazie. W rozdziale 1. pokazano, jak zdefiniować bazę. W tym podrozdziale skoncentrujemy się na tworzeniu tabeli w obrębie istniejącej bazy danych.

W języku SQL instrukcja **`CREATE TABLE`** służy do definiowania tabeli. W przypadku serwera SQL Server 2005 instrukcja ta musi zostać wprowadzona w oknie edytora zapytań.

Ogólna składnia instrukcji **`CREATE TABLE`** jest następująca:

```
CREATE TABLE Nazwa_tabeli
  (nazwa_kolumny typ, nazwa_kolumny typ, .....)
```

W celu zademonstrowania działania instrukcji przedstawimy dwa przykłady.

W ramach pierwszego przykładu utworzymy tabelę o nazwie `Pracownik`, która ma cztery kolumny (atrybuty). Najpierw w oknie edytora zapytań należy wprowadzić poniższą instrukcję (przed rozpoczęciem operacji trzeba się upewnić, czy zaznaczono bazę danych `Student_kurs`; jeśli Czytelnik nie pamięta, w jaki sposób to zrobić, powinien przyjrzeć się rysunkowi 1.16 z rozdziału 1.).

```
CREATE TABLE  Pracownik (personalia    VARCHAR(20),
                         adres         VARCHAR(20),
                         nr_pracownika INT,
                         pensja        SMALLMONEY)
```

[1] „Data Type Performance Tuning Tips for Microsoft SQL Server", *http://www.sql-server-performance.com/datatype.asp*

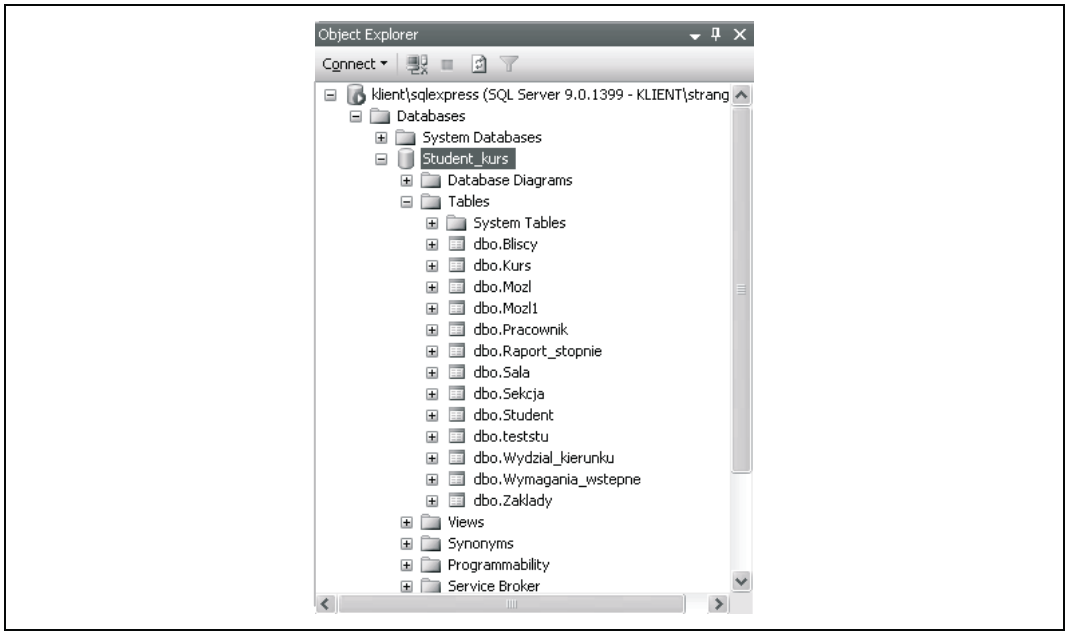
W dalszej kolejności należy wykonać zapytanie.

W efekcie uzyska się następującą odpowiedź:

```
Command(s) completed successfully.
```

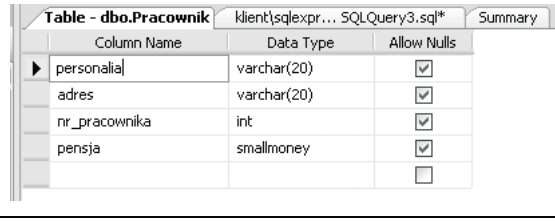
Powyższe zapytanie **CREATE TABLE** utworzyło tabelę (w bazie danych `Student_kurs`) o nazwie `Pracownik` zawierającą 4 kolumny — `personalia`, `adres`, `nr_pracownika` i `pensja`. Typem danych kolumny `personalia` jest `VARCHAR` (znaki o zmiennej długości), dla którego określono maksymalną długość łańcucha wynoszącą 20 znaków. Kolumna `adres` posiada typ danych `VARCHAR` o takiej samej maksymalnej długości jak kolumna `personalia`. Typem danych kolumny `nr_pracownika` jest `INT`, natomiast kolumny `pensja` typ `SMALLMONEY`.

Aby przejrzeć tabelę `Pracownik` bazy danych `Student_kurs`, kolejno należy rozwinąć węzły *Student_kurs* (zlokalizowany w oknie *Object Explorer*) i *Tables*. W efekcie powinno być możliwe zobaczenie tabeli `Pracownik` pokazanej na rysunku 3.1.



Rysunek 3.1. Przeglądanie tabeli Pracownik

W celu zapoznania się z definicją właśnie utworzonej tabeli `Pracownik` należy kliknąć ją prawym przyciskiem myszy i z menu wybrać pozycję *Modify*. Rysunek 3.2 przedstawia definicję tabeli `Pracownik`.



Rysunek 3.2. Definicja tabeli Pracownik

W ramach drugiego przykładu użycia instrukcji **CREATE TABLE** zdefiniujemy tabelę `Personalia` (należy wprowadzić poniższe zapytanie).

```
CREATE TABLE Personalia
   (nazwisko VARCHAR(20))
```

Tabela posiada tylko jedną kolumnę `nazwisko`. Jej typ danych to `VARCHAR`, natomiast maksymalna długość łańcucha zawartego w tej tabeli wynosi 20 znaków.

Umieszczanie wartości w tabeli

W przypadku serwera SQL Server 2005 istnieje kilka metod wstawiania wartości do tabeli za pomocą języka SQL. Zilustrujemy dwie najczęściej wykorzystywane metody. Pierwsza bazuje na instrukcji **INSERT INTO .. VALUES**, natomiast druga stosuje instrukcję **INSERT INTO .. SELECT**.

Zastosowanie instrukcji INSERT INTO .. VALUES

Jednym ze sposobów wstawiania wartości do **jednego** wiersza tabeli jest użycie instrukcji **INSERT INTO** z opcją **VALUES**. Instrukcja **INSERT INTO .. VALUES** wymaga określenia listy kolumn, które muszą być w odpowiedniej kolejności.

Ogólna składnia instrukcji **INSERT INTO .. VALUES** jest następująca:

```
INSERT INTO Nazwa_tabeli
VALUES ('wartość_atrybutu_znakowego', wartość_atrybutu_numerycznego, ...)
```

Najpierw przedstawimy wstawianie danych za pomocą instrukcji **INSERT INTO .. VALUES**. Posłużymy się tabelą `Personalia` utworzoną w poprzednim podrozdziale. W oknie edytora zapytań należy wprowadzić następujące zapytanie:

```
INSERT INTO Personalia
VALUES ('Jan Stachak')
```

Gdzie:

- **INSERT** jest instrukcją języka SQL służącą do wstawiania danych.
- **INTO** jest wymaganym słowem kluczowym.
- `Personalia` jest nazwą istniejącej tabeli.
- **VALUES** jest kolejnym niezbędnym słowem kluczowym.
- `'Jan Stachak'` jest łańcuchem znakowym zgodnym z typem danych `VARCHAR`.

W dalszej kolejności należy kliknąć przycisk *Execute*. Zostanie wyświetlony komunikat informujący o liczbie wierszy dodanych przez zapytanie.

```
(1 row(s) affected)
```

Oto następne zapytanie SQL, które należy wprowadzić:

```
SELECT *
FROM Personalia
```

Efekt jest następujący:

```
nazwisko
--------------------
Jan Stachak

(1 row(s) affected)
```

Instrukcja **INSERT INTO .. VALUES** dodaje do tabeli wiersze (dokładniej mówiąc, są umieszczane na końcu tabeli). A zatem, jeśli ponownie użyje się instrukcji **INSERT INTO .. VALUES** o następującej postaci:

```
INSERT INTO Personalia
VALUES ('Stefan Kowalski')
```

a następnie wykona następujące zapytanie:

```
SELECT *
FROM   Personalia
```

to uzyska się poniższy wynik:

```
nazwisko
--------------------
Jan Stachak
Sudip Kumar

(2 row(s) affected)
```

Jeżeli utworzono tabelę z *n* atrybutami (kolumnami), zwykle w instrukcji **INSERT INTO .. VALUES** znajdzie się *n* wartości podanych zgodnie z kolejnością kolumn w definicji tabeli. Aby na przykład we wcześniej utworzonej tabeli `Pracownik` umieścić wiersz, użyta w tym celu instrukcja **INSERT INTO .. VALUES** musiałaby uwzględniać każdą kolumnę tabeli i wyglądać następująco:

```
INSERT INTO Pracownik
VALUES ('Jan Stachak', 'Ulica 123', 101, 2500)
```

Warto zauważyć, że wprowadzane dane znakowe należy zawrzeć w apostrofach. Dane numeryczne nie wymagają ich (co potwierdzają wartości `101` i `2500`).

Jeśli zastosuje się następujące zapytanie:

```
SELECT *
FROM  Pracownik
```

uzyska się poniższą odpowiedź:

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ---------------------
Jan Stachak          Ulica 123            101           2500,00

(1 row(s) affected)
```

Instrukcja **INSERT** podobna do poniższej jest niepoprawna, ponieważ nie uwzględnia wszystkich czterech kolumn tabeli `Pracownik`.

```
INSERT INTO Pracownik
VALUES ('Jan Stachak', 'Ulica 123')
```

Za pomocą instrukcji **INSERT** można wstawić wiersz, nie uwzględniając wszystkich kolumn tabeli. W tym celu należy wymienić nazwy tylko tych kolumn, w których mają zostać umieszczone dane. Oto przykład:

```
INSERT INTO Pracownik (personalia, adres)
VALUES ('Jan Stachak', 'Ulica 123')
```

W tym przypadku pominięte w instrukcji kolumny będą zawierały wartości puste lub domyślne, które ujrzy się po wykonaniu następującego zapytania:

```
SELECT *
FROM Pracownik
```

Zwróci ono poniższy wynik.

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ----------------------
Jan Stachak          Ulica 123            1O1           25OO,OO
Jan Stachak          Ulica 123            NULL          NULL

(2 row(s) affected)
```

Instrukcja **INSERT** podobna do poniższej jest nieprawidłowa, ponieważ podane w niej wartości nie są zgodne z kolejnością kolumn w definicji tabeli.

```
INSERT INTO Pracownik
VALUES (25OO, 'Jan Stachak', 1O1, 'Ulica 123')
```

Jeśli z jakiegoś powodu dane muszą zostać wprowadzone w powyższej kolejności, instrukcja ta może być poprawiona przez określenie nazw kolumn.

```
INSERT INTO Pracownik (pensja, personalia, nr_pracownika, adres)
VALUES (25OO, 'Jan Stachak', 1O1, 'Ulica 123')
```

Gdy wykona się następujące zapytanie:

```
SELECT *
FROM Pracownik
```

to uzyska się poniższy rezultat:

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ----------------------
Jan Stachak          Ulica 123            1O1           25OO,OO
Jan Stachak          Ulica 123            NULL          NULL
Jan Stachak          Ulica 123            1O1           25OO,OO

(3 row(s) affected)
```

Jeśli nie będą znane wartości kolumn `adres` i `pensja`, właściwie w instrukcji można uwzględnić wartość `NULL`.

```
INSERT INTO Pracownik
VALUES ('Jan Stachak', NULL, 1O1, NULL)
```

Po dodaniu do tabeli czterech wierszy należy wykonać następujące zapytanie:

```
SELECT *
FROM Pracownik
```

Zwróci ono poniższy rezultat:

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ----------------------
Jan Stachak          Ulica 123            1O1           25OO,OO
Jan Stachak          Ulica 123            NULL          NULL
```

```
Jan Stachak          Ulica 123           101          2500,00
Jan Stachak          NULL                101          NULL

(4 row(s) affected)
```

W celu usunięcia wszystkich wierszy tabel `Pracownik` i `Personalia` należy wykonać następujące instrukcje:

```
DELETE FROM Pracownik
DELETE FROM Personalia
```

Instrukcji **DELETE** przyjrzymy się w dalszej części rozdziału.

W pozostałej części rozdziału wypełnimy tabelę `Pracownik` bardziej wartościowymi danymi. Załóżmy, że za pomocą instrukcji **DELETE** usunięto wszystkie testowe wiersze zdefiniowane w poprzednich przykładach. Przyjmijmy ponadto, że przy użyciu instrukcji **INSERT INTO .. VALUES** do tabeli `Pracownik` dodano poprawne dane. W efekcie tabela wygląda następująco:

```
personalia           adres               nr_pracownika pensja
-------------------- ------------------- ------------- ---------------------
Jan Stachak          Kwiatowa 123        101           2500,00
Marek Kowalski       Polna 5             103           3300,00
Anna Szewc           Rumska 11           105           1200,00
Dawid Norek          Barska 23           114           2290,00
Tadeusz Adamski      Nowa 97             95            3309,00

(5 row(s) affected)
```

W jednym oknie edytora zapytań serwera SQL Server 2005 można wprowadzić więcej niż jedną instrukcję **INSERT INTO .. VALUES**.

Zastosowanie instrukcji INSERT INTO .. SELECT

Za pomocą instrukcji **INSERT INTO .. VALUES** do tabeli można wstawić jednocześnie tylko jeden wiersz. Przy użyciu instrukcji **INSERT INTO .. SELECT** za jednym razem można (i zwykle tak się postępuje) umieścić w tabeli **wiele** wierszy.

Ogólna składnia instrukcji **INSERT INTO .. SELECT** jest następująca:

```
INSERT INTO tabela_docelowa(kolumna1, kolumna2, kolumna3, ...)
SELECT ...
```

Najpierw zilustrujemy zastosowanie instrukcji **INSERT INTO .. SELECT**, wypełniając danymi tabelę `Personalia` (została utworzona wcześniej w tym rozdziale, a następnie usunięto z niej wszystkie wiersze za pomocą instrukcji **DELETE FROM** `Personalia`). W celu skopiowania wszystkich personaliów z tabeli `Pracownik` do tabeli `Personalia` należy wykonać następujące zapytanie:

```
INSERT INTO Personalia(nazwisko)
  SELECT  personalia
  FROM    Pracownik
```

Jeśli wprowadzi się poniższą instrukcję:

```
SELECT *
FROM   Personalia
```

Uzyska się następujących pięć wierszy wynikowych:

```
nazwisko
--------------------
Jan Stachak
Marek Kowalski
Anna Szewc
Dawid Norek
Tadeusz Adamski

(5 row(s) affected)
```

Nie musimy kopiować wszystkich personaliów z tabeli `Pracownik` do tabeli `Personalia`. Na przykład instrukcję **`INSERT .. SELECT`** można ograniczyć w następujący sposób:

```
INSERT INTO Personalia(nazwisko)
  SELECT personalia
  FROM   Pracownik
  WHERE  pensja > 2600
```

Po wykonaniu takiego zapytania zostaną zwrócone tylko dwa następujące wiersze tabeli `Personalia`:

```
nazwisko
--------------------
Marek Kowalski
Tadeusz Adamski

(2 row(s) affected)
```

Podobnie jak w przypadku instrukcji **`INSERT INTO .. VALUES`**, gdy tworzy się tabelę o *n* kolumnach, zwykle w instrukcji **`INSERT INTO .. SELECT`** zostanie określonych *n* wartości zgodnie z kolejnością kolumn w definicji tabeli. Ewentualnie trzeba będzie podać w instrukcji nazwy kolumn, które się wstawia. Dla przykładu załóżmy, że utworzono tabelę `Prac1` z trzema kolumnami.

```
Prac1 (adr, pen, nr_prac)
```

Kolumny `adr`, `pen`, `nr_prac` identyfikują odpowiednio adres, pensję i numer pracownika.

Przyjmijmy, że chcemy wypełnić istniejącą pustą tabelę `Prac1` danymi odpowiednich kolumn tabeli `Pracownik`.

Podobnie jak w przypadku instrukcji **`INSERT INTO .. VALUES`**, w instrukcji **`INSERT INTO .. SELECT`** dla każdej kolumny jednej tabeli musi być podana odpowiednia kolumna drugiej tabeli.

Instrukcja **`INSERT INTO .. SELECT`** będzie wyglądać następująco:

```
INSERT INTO Prac1(adr, pen, nr_prac)
  SELECT adres, pensja, nr_pracownika
  FROM   Pracownik
```

Po wykonaniu instrukcji tabela `Prac1` będzie zawierać 5 następujących wierszy:

```
adr                  nr_prac     pen
-------------------- ----------- ---------------------
Kwiatowa 123         101         2500,00
Polna 5              103         3300,00
Rumska 11            105         1200,00
Barska 23            114         2290,00
Nowa 97              95          3309,00

(5 row(s) affected)
```

Jeśli utworzymy tabelę Prac2 z identycznymi kolumnami (lub atrybutami) jak w tabeli Prac1, w celu załadowania danych z tabeli Prac1 w tabeli Prac2 należy skorzystać z poniższej instrukcji **INSERT**.

```
INSERT INTO Prac2
   SELECT *
      FROM Prac1
```

Tabela Prac2 będzie zawierała takie same dane co tabela Prac1. Jest to jedna z metod tworzenia zapasowej tabeli.

Również w tym przypadku trzeba zauważyć, że tabela Prac2 musi istnieć (zdefiniowana z identycznymi kolumnami i typami danych), zanim wypełni się ją danymi za pomocą instrukcji **INSERT INTO .. SELECT**.

Jednakże trzeba ostrzec o jednej rzeczy. Niepoprawna instrukcja **INSERT INTO .. SELECT** może się powieść, gdy typy danych kolumn tabeli określonych za opcją **SELECT** będą zgodne z typami danych kolumn tabeli, do której wstawia się dane. Dla przykładu przyjmijmy, że wykonamy poniższe zapytanie (trzeba pamiętać, że typy danych kolumn pen i nr_prac są numeryczne).

```
INSERT INTO Prac1 (adr, pen, nr_prac)
  SELECT adres, nr_pracownika, pensja
  FROM   Pracownik
```

Ta instrukcja **INSERT** zakończy się powodzeniem, ponieważ typy danych są zgodne. Poniższy wynik uzyska się po wykonaniu ostatniej instrukcji **INSERT**.

```
adr                  nr_prac     pen
-------------------- ----------- --------------------
Kwiatowa 123         2500        101,00
Polna 5              3300        103,00
Rumska 11            1200        105,00
Barska 23            2290        114,00
Nowa 97              3309        95,00

(5 row(s) affected)
```

W kolumnach tabeli Prac1 umieszczono niepoprawne informacje. Zawartość kolumny nr_pracownika tabeli Pracownik wstawiono do kolumny pen tabeli Prac1, natomiast dane kolumny pensja pierwszej tabeli trafiły do kolumny nr_prac drugiej tabeli. W związku z tym, korzystając z instrukcji **INSERT INTO .. SELECT**, trzeba zachować ostrożność i zadbać o zgodność kolumn (atrybutów) wyszczególnianych za instrukcjami **INSERT INTO** i **SELECT**.

Jak można było się domyślić po zapoznaniu z wcześniejszą prezentacją instrukcji **INSERT INTO .. VALUES**, za pomocą instrukcji **INSERT INTO .. SELECT** nie trzeba wstawiać całego wiersza. W przypadku tej instrukcji można uwzględnić mniej kolumn niż wchodzi w skład całego wiersza tabeli Pracownik. Jeśli usunie się z tabeli Prac1 wszystkie wiersze, a następnie wykona instrukcję podobną do następującej:

```
INSERT INTO Prac1 (adr, pen)
  SELECT  adres, pensja
  FROM    Pracownik
```

to instrukcja spowoduje pozostawienie w pominiętej kolumnie nr_prac tabeli Prac1 wartości pustych. Po wykonaniu następującego zapytania:

```
SELECT *
FROM   Prac1
```

otrzyma się poniższy rezultat:

```
adr                  nr_prac     pen
-------------------- ----------- ---------------------
Kwiatowa 123         NULL        2500,00
Polna 5              NULL        3300,00
Rumska 11            NULL        1200,00
Barska 23            NULL        2290,00
Nowa 97              NULL        3309,00

(5 row(s) affected)
```

Wniosek z tego taki, że należy być ostrożnym, gdy stosuje się instrukcję **INSERT INTO .. SELECT**, ponieważ w przeciwieństwie do instrukcji **INSERT INTO .. VALUES** (dodaje jednocześnie jeden wiersz) za jej pomocą prawie zawsze będzie się wstawiało wiele wierszy. Jeśli wystąpi zgodność typów danych, operacja wstawiania zostanie zrealizowana, niezależnie od tego, czy ma sens, czy nie.

Instrukcja UPDATE

Instrukcja **UPDATE** jest kolejną często stosowaną instrukcją. Umożliwia wprowadzanie lub zmienianie wartości danych tabeli. Podobnie jak instrukcja **INSERT INTO .. SELECT**, instrukcja **UPDATE** często jest wykorzystywana do aktualizacji więcej niż jednego wiersza. Aby przekonać się, jak działa ta instrukcja, posłużymy się tabelami zdefiniowanymi wcześniej w tym rozdziale.

Ogólna składnia instrukcji **UPDATE** jest następująca:

```
UPDATE Nazwa_tabeli
SET nazwa_pola ...
```

Jeśli na przykład w tabeli `Prac2` zamierza się dla **wszystkich** wartości kolumny `pen` ustawić wartość zero, można to zrobić przy użyciu jednej instrukcji **UPDATE**.

```
UPDATE Prac2
SET pen = 0
```

Jeżeli wykona się następujące zapytanie:

```
SELECT *
FROM Prac2
```

otrzyma się poniższy rezultat:

```
adr                  nr_prac     pen
-------------------- ----------- ---------------------
Kwiatowa 123         NULL        0,00
Polna 5              NULL        0,00
Rumska 11            NULL        0,00
Barska 23            NULL        0,00
Nowa 97              NULL        0,00

(5 row(s) affected)
```

Powyższa instrukcja **UPDATE** w kolumnie `pen` dla wszystkich wierszy tabeli `Prac2` ustawia wartość zero niezależnie od wcześniejszych wartości. Podobnie jak w przypadku dowolnej instrukcji mającej wpływ na wszystkie wiersze, instrukcja **UPDATE** może być uznana za niebezpieczną i wymagającą ostrożności.

Często przydatne jest uwzględnienie w instrukcji **UPDATE** klauzuli **WHERE**, która pozwala na wybiórcze ustawianie wartości. Jeśli na przykład przyjmiemy, że numery pracowników są unikatowe, za pomocą poniższej instrukcji **UPDATE** można dokonać aktualizacji wiersza tabeli `Pracownik` dla wybranego pracownika.

```
UPDATE Pracownik
SET    pensja = 0
WHERE  nr_prac=101
```

Zapytanie spowoduje następujący wynik:

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ---------------------
Jan Stachak          Kwiatowa 123         101           0,00
Marek Kowalski       Polna 5              103           3300,00
Anna Szewc           Rumska 11            105           1200,00
Dawid Norek          Barska 23            114           2290,00
Tadeusz Adamski      Nowa 97              95            3309,00

(5 row(s) affected)
```

Uaktualniony zostanie tylko wiersz pracownika o numerze 101. Również w tym przypadku trzeba zauważyć, że wartości 101 nie umieszczono w znakach cudzysłowu, ponieważ dla kolumny `nr_pracownika` zdefiniowano numeryczny typ danych `INT`. W znakach cudzysłowu musiałby zostać zawarty dowolny znak lub łańcuch.

Instrukcja ALTER TABLE

W kilku ostatnich podrozdziałach przyjrzeliśmy się temu, jak dodawać, modyfikować i uaktualniać wiersze tabeli za pomocą instrukcji **INSERT** i **UPDATE**. W tym podrozdziale przedstawimy, w jaki sposób dodawać, modyfikować (zmieniać) i usuwać **kolumny** z definicji tabeli przy użyciu instrukcji **ALTER TABLE** języka SQL. Instrukcja ta jest określana mianem instrukcji DDL (*Data Definition Language*), ponieważ zmienia definicję tabeli.

Dodawanie kolumny do tabeli

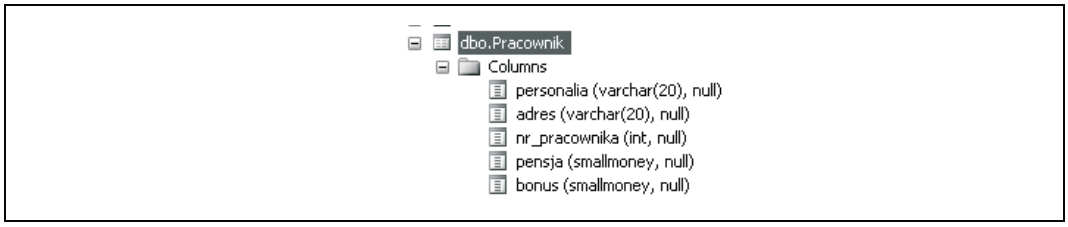
Bez większego problemu do tabeli można dodać kolumny. Ogólna składnia umożliwiającej to instrukcji **ALTER TABLE** jest następująca:

```
ALTER TABLE Nazwa_tabeli
ADD nazwa_kolumny typ
```

W celu dodania do tabeli `Pracownik` na przykład kolumny `bonus` (typu danych `SMALLMONEY`), należy wykonać poniższe zapytanie.

```
ALTER TABLE Pracownik
ADD bonus SMALLMONEY
```

Instrukcja zmienia definicję tabeli `Pracownik` przedstawionej na rysunku 3.3 (w celu uzyskania widoku takiego jak na rysunku, w oknie *Object Explorer* położonym po lewej stronie ekranu należy kliknąć znak plusa znajdujący się obok tabeli `Pracownik`, a następnie kliknąć plus widoczny obok węzła *Columns*).



Rysunek 3.3. Kolumny dodane do tabeli Pracownik

Gdy kolumny są dodawane do istniejących tabel, początkowo będą zawierały wartości puste. Dane można umieszczać w nowej kolumnie za pomocą instrukcji **UPDATE**.

Zmiana typu danych kolumny tabeli

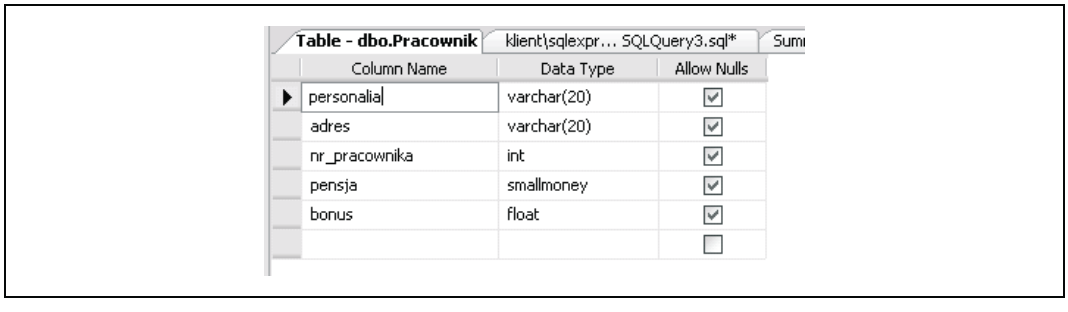
W przypadku serwera SQL Server 2005 można zmienić typ danych kolumny zawierającej dane, przy założeniu że nowy typ obsłuży te dane. Ogólna składnia instrukcji zmieniającej typ danych kolumny tabeli jest następująca:

```
ALTER TABLE Nazwa_tabeli
ALTER COLUMN nazwa_kolumny nowy_typ
```

Aby na przykład zmienić typ danych kolumny `bonus` z `SMALLMONEY` na `FLOAT`, należy zastosować następującą instrukcję:

```
ALTER TABLE PRACOWNIK
ALTER COLUMN bonus FLOAT
```

Zapytanie to wygeneruje definicję tabeli `Pracownik` pokazaną na rysunku 3.4.



Rysunek 3.4. Zmieniony typ danych kolumny bonus tabeli Pracownik

Zanim zobaczy się zmianę wprowadzoną w definicji tabeli, może być konieczne odświeżenie tabeli `Pracownik`. W tym celu prawym przyciskiem myszy należy kliknąć tabelę `Pracownik`, a następnie wybrać pozycję *Refresh*. Po zaznaczeniu tabeli należy wybrać pozycję *Modify*.

Zmiana długości kolumny tabeli

Można zdecydować się na zmianę rozmiaru kolumny tabeli. Zwykle zwiększa się rozmiar kolumny. Dla serwera SQL Server 2005 nie będzie to stanowić problemu, ponieważ większe kolumny obsłużą istniejące dane. Jeśli jednak zamierza się zmniejszyć rozmiar kolumny (jest to rzadkie), czasami serwer zezwoli na to, a czasami nie.

Kiedy serwer SQL Server 2005 bez żadnych problemów umożliwi zredukowanie rozmiaru kolumny?

- Gdy w kolumnie nie ma jeszcze żadnych danych (wszystkie wartości to `NULL`).
- Gdy w dalszym ciągu rozmiar wszystkich danych kolumny nie przekracza wielkości, którą zamierza się dla niej ustawić.

Jeżeli spróbuje się zmniejszyć rozmiar kolumny do wartości, która spowoduje utratę części danych, serwer SQL Server 2005 zwróci błąd i nie pozwoli na to.

Jeśli na przykład wprowadzi się następującą instrukcję **ALTER TABLE**, próbując ustawić rozmiar 5 dla kolumny `personalia` tabeli `Pracownik` (spowoduje to utratę części danych):

```
ALTER TABLE Pracownik
ALTER COLUMN personalia VARCHAR(5)
```

uzyska się poniższy komunikat o błędzie:

```
Msg 8152, Level 16, State 14, Line 1
String or binary data would be truncated.
The statement has been terminated.
```

W momencie sprawdzenia definicji tabeli `Pracownik` okaże się, że rozmiar kolumny `personalia` nie uległ zmianie.

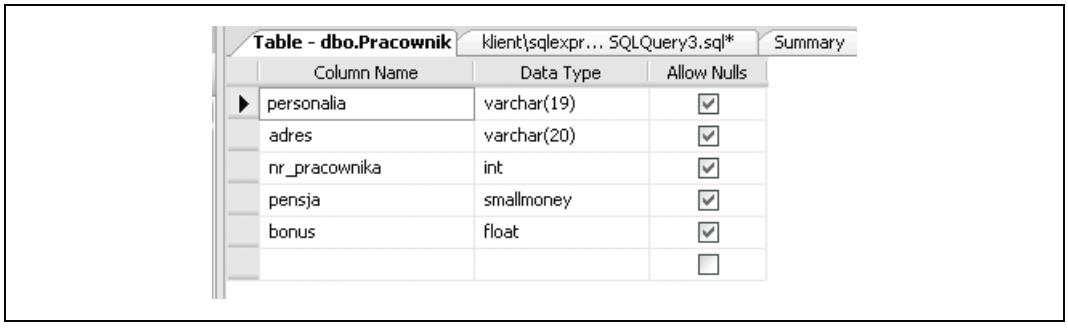
Jeśli jednak wykona się następujące zapytanie:

```
ALTER TABLE Pracownik
ALTER COLUMN personalia VARCHAR(19)
```

otrzyma się poniższy komunikat:

```
Command(s) completed successfully.
```

Po przyjrzeniu się definicji tabeli `Pracownik` stwierdzamy, że aktualnie rozmiar kolumny `personalia` wynosi 19 znaków (rysunek 3.5).



Rysunek 3.5. Zmiana długości kolumny tabeli Pracownik

Jednak zanim będzie można zauważyć zmianę, trzeba będzie odświeżyć tabelę `Pracownik`.

Serwer SQL Server 2005 zezwolił na taką redukcję rozmiaru kolumny `personalia`, ponieważ wszystkie jej dane mają długość nieprzekraczającą 19 znaków.

Przed przejściem do kolejnego punktu dla kolumny `personalia` należy ponownie ustawić rozmiar 20.

Usuwanie kolumny z tabeli

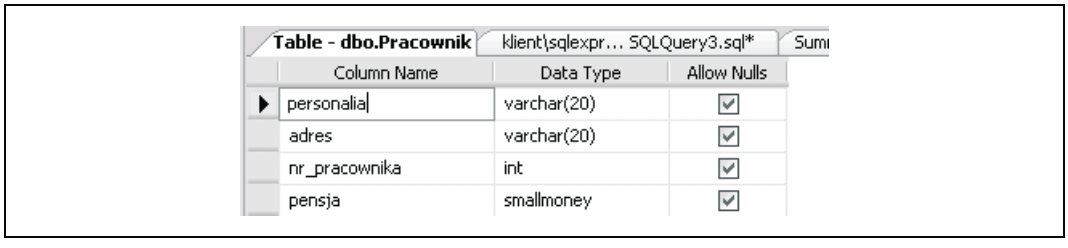
Oto ogólna składnia instrukcji usuwającej kolumnę z tabeli:

```
ALTER TABLE Nazwa_tabeli
DROP COLUMN nazwa_kolumna
```

Aby na przykład usunąć z tabeli `Pracownik` kolumnę `bonus`, należy wykonać następującą instrukcję:

```
ALTER TABLE Pracownik
DROP COLUMN bonus
```

Zapytanie to spowoduje uzyskanie definicji tabeli `Pracownik` przedstawionej na rysunku 3.6, która jest zgodna z oryginalną strukturą tabeli pokazaną na rysunku 3.2.



Rysunek 3.6. Struktura tabeli Pracownik po usunięciu kolumny

Instrukcja **DROP COLUMN** usunie kolumnę nawet wtedy, gdy znajdują się w niej dane. W związku z tym, korzystając z niej, trzeba zachować dużą ostrożność. Jest to kolejna instrukcja, która wpływa na wiele wierszy i musi być używana ze szczególną uwagą.

W kolejnych rozdziałach zostanie omówionych kilka innych zastosowań instrukcji **ALTER TABLE**. Za jej pomocą można na przykład zdefiniować lub zmodyfikować domyślną wartość kolumny, włączyć lub wyłączyć ograniczenie integralności, zarządzać wewnętrzną przestrzenią itp.

Instrukcja DELETE

Wcześniej pokazano w tym rozdziale, że instrukcja **DELETE** może być zastosowana do usunięcia wszystkich wierszy tabeli. Teraz ponownie zajmiemy się bogatą w możliwości instrukcją **DELETE**. Ciągle trzeba pamiętać o tym, że instrukcja **DELETE**, jak już się przekonaliśmy, może mieć wpływ na wiele wierszy. W związku z tym, używając jej, trzeba być ostrożnym. Poniżej podano ogólną składnię instrukcji **DELETE** służącej do usuwania wierszy z tabeli.

```
DELETE FROM Nazwa_tabeli
WHERE (warunek)
```

(warunek) określa, które wiersze tabeli zostaną usunięte. Jak już wcześniej zauważyliśmy, jeśli za klauzulą **WHERE** nie poda się żadnego warunku, zostaną usunięte wszystkie wiersze tabeli.

Ponieważ zakresem działania instrukcji **DELETE** może być objętych wiele wierszy, w jej przypadku należy zachować ostrożność.

Oto przykład użycia instrukcji **DELETE** w przypadku oryginalnej tabeli `Pracownik`.

```
DELETE FROM Pracownik
WHERE pensja < 1500
```

Jeśli wykona się następującą instrukcję:

```
SELECT *
FROM PRACOWNIK
```

to uzyska się poniższy wynik, zawierający trzy wiersze:

```
personalia           adres                nr_pracownika pensja
-------------------- -------------------- ------------- ---------------------
Marek Kowalski       Polna 5              103           3300,00
Dawid Norek          Barska 23            114           2290,00
Tadeusz Adamski      Nowa 97              95            3309,00

(3 row(s) affected)
```

Usuwanie tabeli

Ogólna składnia instrukcji usuwającej całą tabelę i jej zawartość jest następująca:

```
DROP TABLE Nazwa_tabeli
```

Aby na przykład z bazy danych usunąć tabelę `Personalia`, należy wykonać poniższą instrukcję.

```
DROP TABLE Personalia
```

Zdarzają się sytuacje, w których wskazane jest wyczyszczenie wszystkich danych tabeli, i takie, że cała tabela powinna zostać usunięta z bazy. Po usunięciu tabela nie będzie już dostępna. Jej definicja zostanie usunięta z bazy danych. Jednak gdy dane zostaną usunięte z tabeli za pomocą instrukcji **DELETE** (być może z uwzględnieniem warunku klauzuli **WHERE**), tabela może być ponownie nimi wypełniona. Wynika to stąd, że z tabeli usunięto dane, lecz jej definicja pozostała nienaruszona.

Podsumowanie

W rozdziale zajęliśmy się podstawowymi operacjami wykonywanymi na tabelach. Wyjaśniliśmy, w jaki sposób tworzyć tabele, umieszczać w nich dane, aktualizować je, dodawać i usuwać kolumny tabel, modyfikować typ danych i rozmiar kolumn, a także usuwać całe tabele. Przedstawiliśmy również podstawowe typy danych dostępne w serwerze SQL Server 2005.

Pytania kontrolne

1. Instrukcja **INSERT INTO .. VALUES** wstawi wartości do _____ tabeli.
2. Czy kolejność kolumn podanych za instrukcją **INSERT** musi być taka sama jak kolejność kolumn tabeli, gdy wartości umieszcza się w tabeli za pomocą instrukcji **INSERT INTO .. VALUES**?
3. Czy kolejność kolumn podanych za instrukcją **INSERT** musi być taka sama jak kolejność kolumn tabeli, gdy wartości umieszcza się w tabeli za pomocą instrukcji **INSERT INTO .. SELECT**?

4. Kiedy stosuje się instrukcję **INSERT INTO .. SELECT**, a kiedy instrukcję **INSERT INTO .. VALUES**? Dla każdej instrukcji należy podać przykład użycia.
5. Do czego służy instrukcja **UPDATE**?
6. Czy można zmienić typ danych kolumny tabeli już po jej utworzeniu? Jeśli tak, jakiej należy użyć instrukcji?
7. Czy serwer SQL Server 2005 umożliwia zredukowanie rozmiaru kolumny?
8. Jakie całkowitoliczbowe typy danych są oferowane przez serwer SQL Server 2005?
9. Jaka w przypadku serwera SQL Server 2005 jest domyślna wartość całkowitoliczbowego typu danych?
10. Jakie dziesiętne typy danych są oferowane przez serwer SQL Server 2005?
11. Jaka jest różnica między typem danych `CHAR` i `VARCHAR`?
12. Czy serwer SQL Server traktuje kolumnę typu danych `CHAR` jako posiadającą zmienną czy stałą długość? Czy inne serwery bazodanowe wykorzystujące język SQL traktują tego typu kolumny w taki sam sposób?
13. Jaki typ danych będzie najlepszą propozycją, gdy w kolumnie zamierza się przechowywać bardzo wiele wartości pustych?
14. Co na początku będą zawierać kolumny dodawane do istniejących tabel?
15. Jakiej instrukcji należy użyć w przypadku serwera SQL Server w celu dodania kolumny do tabeli?
16. Jaki typ danych w przypadku serwera SQL Server jest wykorzystywany do przechowywania dużych obiektów?
17. Jaki numeryczny typ danych będzie odpowiedni, gdy nie ma potrzeby przechowywania miejsc dziesiętnych?
18. Jaki typ danych będzie właściwy, gdy trzeba przechowywać miejsca dziesiętne, lecz nie są istotne błędy zaokrąglania?
19. Czy podczas definiowania kolumny, która będzie używana w roli klucza podstawowego, powinno się zastosować typ danych `FLOAT`?

Ćwiczenia

Jeśli nie zaznaczono inaczej, w celu udzielenia odpowiedzi na poniższe pytania należy posłużyć się bazą danych `Student_kurs`. Ponadto, wyświetlając wynik, należy użyć odpowiednich nagłówków kolumn.

1. Należy utworzyć tabelę `Klient` zawierającą numer klienta (łańcuch o stałej długości równej 3 znakom), jego adres (łańcuch o zmiennej długości wynoszącej maksymalnie 20 znaków) i saldo.
   a. Za pomocą instrukcji **INSERT INTO .. VALUES** należy w tabeli umieścić wartości. Należy zastosować postać instrukcji, która wymaga podania wartości dla każdej kolumny. W związku z tym, gdy istnieją kolumny z numerem klienta, jego adresem i saldem, przy użyciu instrukcji **INSERT INTO .. VALUES** trzeba będzie wstawić trzy wartości.

b. Do tabeli należy dodać przynajmniej 5 wierszy, w przypadku których numery klientów zawierają się w przedziale od 101 do 105, natomiast saldo przyjmuje wartości z zakresu od 200 do 2000.

c. Za pomocą prostej instrukcji **SELECT** należy wyświetlić zawartość tabeli.

d. Dla klientów o numerach 103 i 104 należy wyświetlić saldo.

e. Do tabeli `Klient` należy dodać klienta o numerze 90.

f. Przy użyciu klauzuli **ORDER BY** instrukcji **SELECT** należy wyświetlić listę klientów posortowanych według salda (od największego do najmniejszego). Jako wynik powinno się uzyskać 5 wierszy lub tyle, ile utworzono.

2. Korzystając z tabeli `Student` (zawarta w bazie danych `Student_kurs`), w malejącej kolejności oznaczeń liczbowych grup należy wyświetlić imiona, grupy i kierunki dla studentów pierwszego i drugiego roku (`grupa <= 2`).

3. Używając tabeli `Klient`, w malejącej kolejności należy wyświetlić tylko salda klientów, w przypadku których saldo jest większe od 400 (w razie potrzeby można zastosować jakąś inną stałą lub zależność, taką jak `saldo <= 600`). Uzyskane wyniki będą zależne od danych.

4. Należy utworzyć kolejne dwie tabele tego samego typu danych, co tabela `Klient`, lecz pozbawione adresów klientów. Tabelom należy nadać nazwy `Klient1` i `Klient2`. Dla kolumny numerów klientów należy użyć nazwy `kl_num`, natomiast dla kolumny salda nazwy `saldo`. Tabele należy wypełnić danymi pobranymi z tabeli `Klient`, z wyłączeniem jednego wiersza. W instrukcji **INSERT INTO .. SELECT** należy podać odpowiednie kolumny i zastosować właściwy warunek klauzuli **WHERE**.

   a. Należy wyświetlić zawartość uzyskanych tabel.

5. Dodając do tabeli `Klient1` kolumnę `data_otwarcia` typu danych `DATETIME`, należy zmodyfikować tabelę. Należy zapoznać się z definicją tabeli `Klient1`.

   a. Za pomocą instrukcji **INSERT INTO .. VALUES** należy dodać więcej danych do tabeli `Klient1`.

   Po wykonaniu każdej poniższej operacji należy wyświetlić zawartość tabeli.

   b. W polu kolumny `data_otwarcia`, we wszystkich wierszach, należy ustawić wartość `'01-01-06'`.

   c. Wszystkie salda należy wyzerować.

   d. Dla jednego z wierszy w polu kolumny `data_otwarcia` należy ustawić wartość `'10-21-06'`.

   e. Typ danych kolumny `saldo` tabeli `Klient1` należy zmienić na `FLOAT`, a następnie wyświetlić definicję tabeli. Po ustawieniu w jednym z wierszy salda o wartości 888,88 należy wyświetlić zawartość tabeli.

   f. Należy spróbować zmienić typ danych kolumny `saldo` na `INT`. Czy serwer SQL Server na to pozwoli?

   g. Z tabeli `Klient1` należy usunąć kolumnę `data_otwarcia`.

   h. Po ukończeniu ćwiczenia (trzeba być tego całkowicie pewnym) należy usunąć tabele `Klient`, `Klient1` i `Klient2`.