Bazy danych – Semestr 2
Zajęcia nr 3

Wstęp do pisania skryptów

Wprowadzenie do instrukcji Transact-SQL

- Najprostszy sposób korzystania z okna Query polega na bezpośrednim wpisywaniu w nim instrukcji SQL
- Okno query oferuje dodatkowe udogodnienia polegające na kolorowym zapisie wprowadzanych instrukcji języka Transact-SQL

Kolor	Znaczenie
Niebieski	Słowo kluczowe
Ciemnozielony	Komentarz
Ciemnoczerwony	Procedura przechowywana
Szary	Operator
Zielony	Tabela systemowa
Purpurowy	Funkcja systemowa
Czerwony	Ciąg znaków

Skrypt SQL

- Skrypt (ang. scripts) jest zestawem instrukcji języka Transact-SQL przechowywanym w pliku.
- Skrypty są najczęściej używane do przechowywania stałego zbioru poleceń używanego do tworzenia i wypełniania baz obiektów. Ponieważ skrypty są przechowywane w plikach tekstowych a nie w bazie danych, mogą być używane do ponownego tworzenia takich samych baz danych na różnych serwerach.
- Instrukcje SQL w skrypcie są pogrupowane w tzw. „wsady” (paczki) (ang. batches).
- Skrypt może zawierać jeden lub kilka wsadów, zaś każdy wsad może składać się z jednej lub kilku instrukcji SQL.
- W skrypcie zawierającym kilka wsadów są one oddzielone poleceniami **GO**.
- Wyrażenia tworzące paczkę wykonywane są jednorazowo w jednym czasie. W przypadku wystąpienia błędu kompilacji, żadne z wyrażeń w paczce nie zostanie wykonane.

Przykładowy Skrypt SQL

Przykład skryptu złożonego z trzech paczek:

```
CREATE DATBASE KASETY_19
GO -- wykonanie paczki, tworzenie bazy „KASETY_19”
USE KASETY_19
GO   -- wykonanie paczki, wybór bazy (ustawienie się domyślne) „KASETY_19”
CREATE TABLE osoby_test
  (
    ido          int    PRIMARY KEY,
    imie         char(10),
    nazwisko char(13)
  )
GO           -- wykonanie paczki, stwórz tabelę osoby_test w bazie „KASETY_19”
INSERT into osoby_test values (1,'jan', 'kowal')
GO           -- wykonanie paczki, wstaw dane do tabeli osoby_test
```

Komentarze

- Tak jak programy pisane w proceduralnych językach programowania, programy języka Transact-SQL zyskują na czytelności poprzez dodanie do kodu komentarzy opisujących role poszczególnych zmiennych czy funkcję bardziej skomplikowanych fragmentów programu.
- Komentarze są ignorowanymi przez kompilator ciągami znaków wyróżnionymi na jeden z dwóch sposobów:
  - Za pomocą dwóch znaków myślnika (— —). Kompilator ignoruje znajdujące się po prawej stronie myślników znaki.
  - Przykład:

```
-- komentarz tekstowy
```

  - Za pomocą znaków /* */. Kompilator ignoruje wszystkie znaki (z wyjątkiem dyrektywy GO) znajdujące się pomiędzy wspomnianymi znacznikami.
  - Przykład

```
/*
komentarz
Komentarz
*/
```

Instrukcje DDL

- Instrukcje DDL (ang. Data Definition Language — język definiowania danych) tworzące i modyfikujące obiekty bazodanowe.
- Do instrukcji DDL zalicza się polecenia:
  - CREATE typ_obiektu nazwa_obiektu
  - ALTER typ_obiektu nazwa_obiektu
  - DROP typ_obiektu nazwa_obiektu
- Domyślne zasady bezpieczeństwa SQL Severa umożliwiają wykonywanie instrukcji DDL jedynie:
  - członkom ról serwera (sysadmin, dbcreator)
  - oraz ról bazy danych (db_owner, db_ddladmin)
- Ponieważ użytkownik, który utworzył obiekt, a nie należał do grupy sysadmin, staje się właścicielem obiektu (obiekty utworzone przez administratorów stają się własnością specjalnego użytkownika dbo), nie należy, o ile jest to tylko możliwe, przypisywać do wymienionych ról kont użytkowników

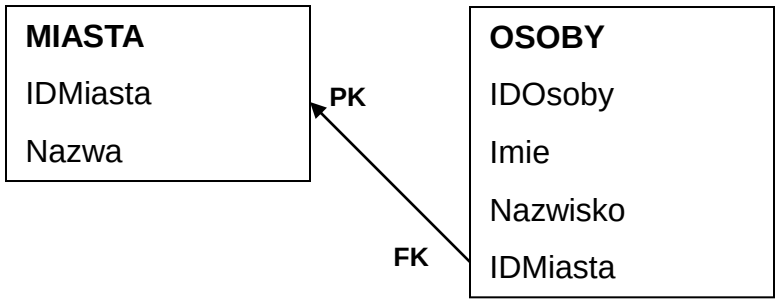
Tworzenie tabeli: Create Table

- Przy pomocy wyrażenia **CREATE TABLE** wprowadzamy nazwę tablicy (maksymalnie 128 znaków dla zwykłej tablicy lub 116 znaków dla tablicy tymczasowej, której nazwa musi zaczynać się od #) oraz definiujemy jedną lub więcej kolumn
- Nazwy kolumn (do 128 znaków) muszą być unikalne w ramach tablicy
- Definicja każdej kolumny musi zawierać jej nazwę oraz typ danych
- Aby wymusić wprowadzanie danych do kolumny używamy zwrotu NOT NULL
- Wartość domyślną dla kolumny można wprowadzić za pomocą zwrotu DEFAULT wartość
- Kolumna zawierająca dane numeryczne dokładne może mieć automatycznie wprowadzane unikalne, narastające wartości, jeżeli użyty zostanie zwrot IDENTITY [(liczba początkowa, skok)] np. (1,1)
- Kolumna może być oznaczona jako klucz główny (PRIMARY KEY) lub o niepowtarzających się wartościach (UNIQUE)
- Kolumna może być kluczem obcym ([FOREIGN KEY] REFERENCES), którego wartości muszą występować we wskazanej unikalnej kolumnie innej tabeli

Tworzenie powiązania między 2 tabelami Sposób 1

- Kolejność tworzonych tabel w tej metodzie **nie jest istotna**

```
-- ****** Tworzenie Tabeli: MIASTA
-- ------------------------------------------------
CREATE TABLE MIASTA
    (       IDMiasta  int          PRIMARY KEY,
            Nazwa     char (30)
    )
GO
-- ******* Tworzenie tabeli: OSOBY
-- ------------------------------------------------
CREATE TABLE OSOBY
    (       IDOsoby   int          PRIMARY KEY,
            Imie      char (20),
            Nazwisko  char (20),
            IDMiasta  int
    )
GO
-- ****** Tworzenie powiązań referencyjnych „Relacji" (powiązań między
    tabelami -ALTER (polski: zmień)
ALTER TABLE OSOBY ADD
    CONSTRAINT [FK_OSOBY_MIASTA] FOREIGN KEY
    (IDMiasta) REFERENCES Miasta(IDMiasta)
GO
```



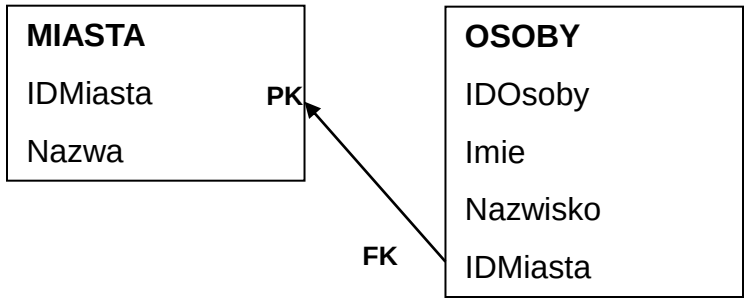
Zmiana schematu tabeli: Alter Table

- Polecenie ALTER TABLE umożliwia:
  - dodanie nowej kolumny
  - modyfikację kolumny
  - bądź usunięcie istniejącej kolumny

Tworzenie powiązania między 2 tabelami Sposób 2

- Kolejność tworzonych tabel w tej metodzie **jest istotna**

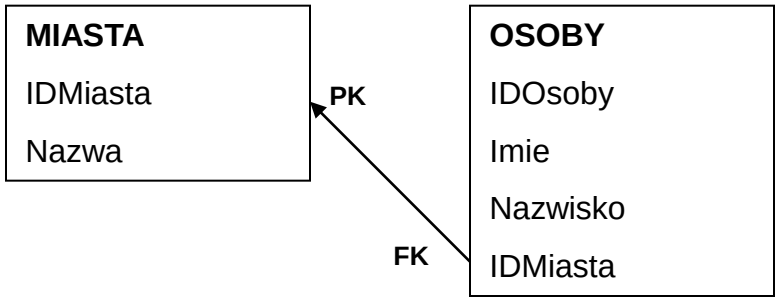
```
-- ****** Tworzenie Tabeli: MIASTA
-- ------------------------------------------------
CREATE TABLE MIASTA
    (       IDMiasta  int           PRIMARY KEY,
            Nazwa     char (30)
    )
GO
-- ******* Tworzenie tabeli: OSOBY
-- ------------------------------------------------
CREATE TABLE OSOBY
    (       IDOsoby   int           PRIMARY KEY,
            Imie                    char (20),
            Nazwisko                char (20),
            IDMiasta                int     REFERENCES Miasta(IDMiasta)
    )
GO
```



Tworzenie powiązania między 2 tabelami Sposób 3

- Kolejność tworzonych tabel w tej metodzie **jest istotna**
- nazwa powiązania jest definiowana w skrypcie np. FK_OSOBY_MIASTA

```
-- ****** Tworzenie Tabeli: MIASTA
-- ------------------------------------------------
CREATE TABLE MIASTA
    (       IDMiasta    int             PRIMARY KEY,
            Nazwa       char (30)
    )
GO
-- ******* Tworzenie tabeli: OSOBY
-- ------------------------------------------------
CREATE TABLE OSOBY
    (       IDOsoby     int             PRIMARY KEY,
            Imie                        char (20),
            Nazwisko                    char (20),
            IDMiasta                    int
            CONSTRAINT [FK_osoby_miasta] FOREIGN KEY (IDMiasta) references MIASTA(IDMiasta)
    )
GO
```



- select * from dbo.sysobjects
- select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS

Sprawdzenie istnienia tabeli (Skrypt SQL)

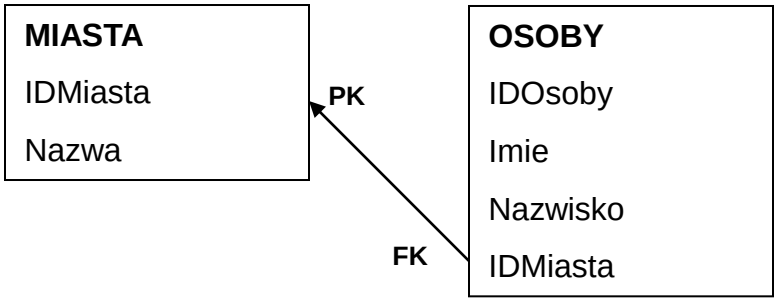
```
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'MIASTA'))
    BEGIN
        PRINT '   Tabela MIASTA istnieje w BD KASETY!'
    END
ELSE
    BEGIN
        PRINT '   Tworze tabele MIASTA BD KASETY'
        CREATE TABLE MIASTA
                (
                IDMiasta  int  PRIMARY KEY,
                Nazwa     char (30)
                )
    END
GO
```

Tabela systemowa

Sysobjects – BAZA (nazwa bazy) -> VIEWS -> System Views -> Sysobjects

Skrypt do tworzenia 2 tabel + powiązanie

```
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'MIASTA'))
	BEGIN
		PRINT '  Tabela MIASTA istnieje w BD KASETY!'
	END
ELSE
	BEGIN
		PRINT '  Tworze tabele MIASTA BD KASETY'
		CREATE TABLE MIASTA
		(
			IDMiasta	int		PRIMARY KEY,
			Nazwa		char (30)
		)
	END
GO
-- ----------------------------Tworzenie tabeli: OSOBY
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'OSOBY'))
	BEGIN
		PRINT '  Tabela OSOBY istnieje w BD KASETY!'
	END
ELSE
	BEGIN
		PRINT '  Tworze tabele OSOBY BD KASETY'
		CREATE TABLE OSOBY
		(		IDOsoby		int		PRIMARY KEY,
			Imie		char (20),
			Nazwisko	char (20),
			IDMiasta	int
		)
	END
GO
-- --------- Tworzenie powiazania miedzy tabelami -ALTER (polski: zmień)
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'FK_OSOBY_MIASTA'))
	BEGIN
		PRINT '  Powi'zanie OSOBY-MIASTA istnieje w BD KASETY!'
	END
ELSE
	BEGIN
		PRINT '  Tworze Powiazanie MIASTA-OSOBY BD KASETY'
		ALTER TABLE OSOBY ADD
			CONSTRAINT [FK_OSOBY_MIASTA] FOREIGN KEY
		(IDMiasta) REFERENCES Miasta(IDMiasta)
	END
GO
```



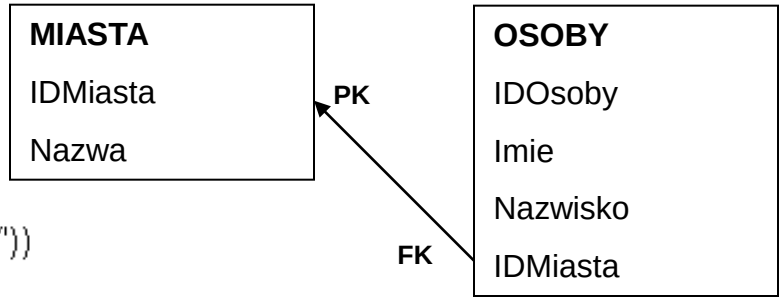
Usuwanie tabeli: Drop Table

- Polecenie **DROP TABLE** usuwa:
  - tabelę i wszystkie zawarte w niej dane z bazy danych
  - wszystkie zdefiniowane na niej indeksy
  - oraz wszystkie związane z nią uprawnienia dostępu.
- Nie można usunąć tabeli wskazanej jako odniesienie w kluczu obcym w innej tabeli.
- Najpierw należy usunąć tamtą tabelę lub ograniczenie FOREIGN KEY.
- Usunięcie wszystkich danych z tabeli nie powoduje usunięcia tabeli

Tworzenie skryptu do usuwania tabel

1) Usuwanie powiązań między tabelami [FK_OSOBY_MIASTA]

```
-- ------------------------------------------------
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'FK_OSOBY_MIASTA'))
      BEGIN
        PRINT '   Usuwam powiązanie [FK_OSOBY_MIASTA] w bazie danych KASETY'
        ALTER TABLE OSOBY DROP
                              CONSTRAINT [FK_OSOBY_MIASTA]
      END
ELSE
       PRINT ' UWAGA!- Brak powiązania [FK_OSOBY_MIASTA] w bazie danych KASETY'
GO
```



2) Usuwanie tabeli OSOBY

```
-- ------------------------------------------------
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'OSOBY'))
      BEGIN
              DROP table OSOBY
              PRINT '   Usuwam Tabelę OSOBY z bazy danych KASETY'
      END
ELSE
              PRINT '   UWAGA !          - Brak Tabeli OSOBY w bazie danych KASETY'
GO
```