

# Bazy danych – Semestr 2

## Zajęcia nr 6

Pisanie skryptów  
kontynuacja

# Zakres zajęć

- Pisanie skryptu do tworzenia bazy danych „Wypożyczalnia filmów” – **kontynuacja**
- Wyzwalacze (triggery) – dokończenie (*INSTEAD OF*)
- Procedury składowane
- Stosowanie zmiennych
- Instrukcja warunkowa IF
- Instrukcja warunkowa CASE
- Instrukcja warunkowa WHILE
- Kursory - wprowadzenie

# Procedury wyzwalane (ang.Triggers)

- Tabela **KomunikatyWyzwalarczy** zbiera informacje o działaniach użytkowników na zasobach bazy danych (wprowadzanie, modyfikacja i usuwanie wierszy danych w wybranych tabelach tej bazy)
- Czy tabela jest bezpieczna (zabezpieczona)?
- Czy nie powinno być tak, że użytkownik może wprowadzać tam dane (z wykorzystaniem innych triggerów) ale nie powinien mieć możliwości usuwania i modyfikacji tam wierszy już istniejących ?
- Jak zablokować użytkownikom operacje usuwania i modyfikacji wpisów w tabeli **KomunikatyWyzwalaczy**.
- Do tego celu wykorzystamy wyzwalacz **INSTEAD OF** przypisany do zdarzeń UPDATE i DELETE. Spowoduje on, że przy każdej próbie zmodyfikowania lub usunięcia wpisu w tabeli **KomunikatyWyzwalaczy**, zamiast wykonywanej operacji wykona się „nasz wyzwalacz”, którego jedyną funkcjonalnością jest wygenerowanie komunikatu o błędzie.
- Przed utworzeniem triggera usuńmy wiersz danych z tabeli **KomunikatyWyzwalarczy**
- .....udało się?

# Procedury wyzwalane (ang.Triggers) -1

.... Niestety tak ! (Utraciliśmy kontrolę nad modyfikacją bazy „Wypożyczalnia filmów”  
Składnia triggera

```
CREATE TRIGGER tr_blokada_modyfikacji  
ON KOMUNIKATYWYZWALACZY  
INSTEAD OF UPDATE,DELETE  
AS  
RAISERROR('Edycja i usuwanie wpisów są zabronione',16,1)
```

Poziom błędu, stan błędu



Po utworzeniu triggera usuńmy/zmodyfikujmy wiersz danych w tabeli  
***KomunikatyWyzwalarczy***

.....udało się ? .... Nie.. Sukces !!!!!!!!!!!!!!!!!!!!!

RAISERROR(msg\_str, waga, stan) (podobnie jak PRINT) służy do tworzenia komunikatów – ***UWAGA: stosowano do wersji 2012***

- **Msg\_str** – tekst komunikatu
- **Waga** – waga komunikatu (0-10 informacyjny, 11-16 – błędy, które może naprawić użytkownik, 17-18 – poważniejsze błędy, 19 – wewnętrzny błąd związany z zasobami, 20-25 – błędy krytyczne
- **stan** komunikatu dodatkowa informacja o komunikacie, np. nr wiersza, w którym pojawił się błąd

# Procedury wyzwalane (ang.Triggers) - 2

## Składnia triggera

```
CREATE TRIGGER tr_blokada_modyfikacji  
ON KOMUNIKATYWYZWALACZY  
INSTEAD OF UPDATE,DELETE  
AS  
THROW 60001, 'Edycja i usuwanie wpisów sa zabronione',10
```

Po utworzeniu triggera usuńmy/zmodyfikujmy wiersz danych w tabeli

### ***KomunikatyWyzwalarczy***

.....udało się ? .... Nie.. Sukces !!!!!!!!!!!!!!!!

THROW(error\_number, error\_message, error\_state) (podobnie jak PRINT) służy do tworzenia komunikatów

- **Error\_number** - numer błędu (wartość INT nie mniejsza od 50000)
- **Error\_message** – tekst komunikatu
- **Error\_state** stan komunikatu stan, wartość z przedziału 0 – 255

Przykład użycia:

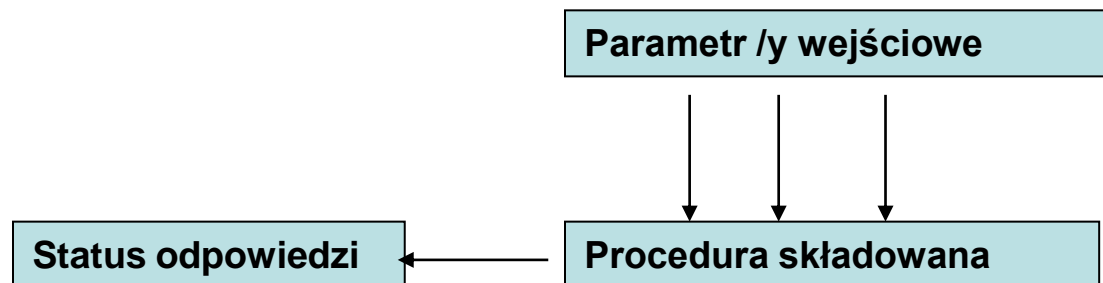
```
THROW 50001, 'Własny komunikat błędu', 3;
```

 Messages

Msg 50001, Level 16, State 3, Line 1  
Własny komunikat błędu

# Procedury składowane

- **Procedury składowane** przechowywane są w bazie w postaci instrukcji Transact-SQL i wykonywane przez serwer bazy danych.
- Można je wywoływać:
  - samodzielnie,
  - poprzez inne procedury przechowywane,
  - poprzez procedury wyzwalane,
  - a także z poziomu aplikacji klienckich.
- Mają możliwość pobierania parametrów i zwracania wyniku (co zwiększa ich funkcjonalność).
- Dużą zaletą ich stosowania jest podniesienie wydajności serwera bazy danych. Używając procedury składowanej zamiast zapytania SQL, np. w skrypcie na stronie WWW, przy wielu jednoczesnych wywołaniach tej strony odciążamy serwer - wykonujemy tylko jedną analizę kodu zamiast wielu kompilacji powtarzającego się zapytania SQL.



# Procedury składowane

## Przykład

### 1. Procedura dodająca nowy kraj do bazy danych

```
CREATE PROCEDURE zapisz_nowy_kraj
    @idkraj int,
    @krajprod char(15)
AS
IF EXISTS (SELECT * FROM kraj WHERE idkraj=@idkraj)
    RETURN 1
IF NOT EXISTS (SELECT * FROM kraj WHERE krajprod=@krajprod)
    RETURN 2
BEGIN TRANSACTION
    INSERT INTO kraj(idkraj,krajprod)
        VALUES (@idkraj,@krajprod)
    IF @@ERROR <> 0
        GOTO BLAD
COMMIT TRANSACTION
RETURN 0
BLAD:
    ROLLBACK TRANSACTION
    RETURN 2
GO
```

Parametry wejściowe

Status odpowiedzi

# Procedury składowane

## Przykład wywołania

```
EXEC zapisz_nowy_kraj 16,'RRRS'
```

*Powyższa metoda nie informuje użytkownika (brak zwracanego stanu wykonania) o tym czy procedura wykonała się poprawnie czy nie*

## Kontrola poprawności wykonania.

```
DECLARE @return_status int  
EXEC @return_status = zapisz_nowy_kraj 25,'Malta'  
SELECT 'Return Status' = @return_status  
GO
```

*Powyższa metoda informuje użytkownika o tym (zwraca status) czy procedura wykonała się poprawnie (np. status = 0) czy nie (gdy niepoprawnie zaprogramowane w procedurze statusy <błędy>)*



# Stosowanie zmiennych(1)

**Zmienne mogą być lokalne** (znak @) lub **globalne** (znaki @@)

Np.

```
DECLARE @zmienna_lokalna typ_danych
```

```
DECLARE @zmienna1 int, @zmienna2 int
```

Po utworzeniu zmiennej jej początkowa wartością jest NULL

**Przypisanie wartości:**

```
SET @MojaZmienna ='witajcie'
```

```
SELECT @MojaZmienna='witajcie'
```

```
SELECT @MaxCena = MAX(cena) FROM FILMY
```

# Stosowanie zmiennych(2)

```
declare @moje nvarchar(50), @MAXCENA decimal(6,2)
```

```
select @moje
```

```
select @moje = 'witajcie11'
```

```
select @moje
```

## **Przypisanie wartości:**

```
SET @MojaZmienna ='witajcie'
```

```
SELECT @MojaZmienna
```

```
SELECT @MaxCena = MAX(cena) FROM FILMY
```

```
-- wyświetlenie (pokazanie) wartości
```

```
SELECT @MaxCena
```

# Stosowanie zmiennych(3)

## **ZMIENNE TABULARNE (tabelarne)**

### **Deklaracja**

```
declare @lokalnatab TABLE (tytul char(50) ,cena decimal(6,2))
```

### **Utworzenie wierszy**

```
INSERT INTO @lokalnatab SELECT tytul,cena FROM FILMY
```

### **Wyświetlenie wyników**

```
SELECT * from @lokalnatab
```

### **Wyświetl informacje o serwerze przy pomocy zmiennych globalnych**

```
select @@servername, @@version, @@language
```

# Instrukcja warunkowa IF

## Przykład

Napisać funkcję pokazującą informację w przypadku gdy:

- mężczyźni jest więcej od kobiet (w tabeli klienci) wypisany będzie komunikat „MĘŻCZYŹN JEST WIĘCEJ O” i podana liczba
- kobiet jest więcej od mężczyzn (w tabeli klienci) wypisany będzie komunikat „Kobiet jest więcej O” i podana liczba

```
declare @m int, @k int
select @m=count(*) from Klienci where plec='M'
select @k=count(*) from Klienci where plec='K'
select 'kobiety', @k
select 'mężczyźni', @m
```

```
IF @m > @k
BEGIN
    PRINT 'MĘŻCZYŹN JEST WIĘCEJ'
    SELECT 'MĘŻCZYŹN JEST WIĘCEJ O', @m-@k
END
ELSE
BEGIN
    PRINT 'Kobiet jest więcej'
    SELECT 'Kobiet jest więcej O', @k-@m
END
```

# Instrukcja warunkowa CASE

## Przykład

Napisać select pokazujący informacje nazwisko, imię i płeć (klientów) w przypadku gdy:

- Płeć = 'M' wypisze 'mężczyzna'
- Płeć = 'K' wypisze 'kobieta'

```
SELECT nazwisko, imię,  
CASE plec  
  WHEN 'K' THEN 'Kobieta'  
  WHEN 'M' THEN 'Mężczyzna'  
  ELSE 'Ktoś'  
END AS Płeć  
FROM Klienci  
Order by nazwisko
```

# Pętla WHILE

## Przykład

Napisać funkcję wykonującą się w pętli (x? – krotnie)

```
DECLARE @LICZNIK INT
SET @LICZNIK = 1
WHILE @LICZNIK < 11
BEGIN
    PRINT @LICZNIK
    SET @LICZNIK = @LICZNIK+1
END
```

```
DECLARE @LICZNIK INT
SET @LICZNIK = 1
WHILE @LICZNIK < 11
BEGIN
    SET @LICZNIK = @LICZNIK + 1
    IF (@LICZNIK % 2) = 1 CONTINUE
    PRINT @LICZNIK
END
```

# Kursory - wprowadzenie

**Kursor** jest obiektem wskazującym określony wiersz w zestawie.

W zależności od charakteru kursora, przy jego użyciu można przemieszczać się pomiędzy wierszami zestawu i aktualizować lub usuwać dane.

**Składnia instrukcji: *DECLARE CURSOR***

**DECLARE** *nazwa* **CURSOR**

*[widoczność]*

*[przewijanie]*

*[typ]*

*[blokada]*

**[TYPE\_WARNING]**

**FOR** *instrukcja selekcji*

**[FOR UPDATE [OF *nazwa\_kolumny*]]**

- **Widoczność:** LOKAL lub GLOBAL
- **Przewijanie:** FORWARD\_ONLY (tylko do przodu)  
                  SCROLL (obydwa kierunki)
- **Typ:**            STATIC     - statyczny  
                  KEYSET    - kluczowe  
                  DYNAMIC   - dynamiczne  
                  FAST\_FORWARD - błyskawiczne
- **Blokada:**        READ\_ONLY - nie można modyfikować danych

# Kursory

## Otwieranie kursora: **OPEN**

Deklaracja tworzy obiekt kursora nie tworzy zestawu rekordów który będzie przetwarzany przy jego użyciu.

**OPEN** *cursor\_nazwa*

## Zamykanie kursora: **CLOSE**

Po zakończeniu korzystania z kursora należy go zamknąć. Instrukcja CLOSE zwalnia zasoby systemowe używane do obsługi zestawu kursora.

**CLOSE** *Cursor\_nazwa*

## Zwalnianie kursora: **DEALLOCATE**

Usuwanie identyfikatora kursora

**DEALLOCATE** *cursor\_nazwa*



# Kursory

## Manipulowanie wierszami za pomocą kursora:

Pobierz określony wiersz z zestawu kursora:

**FETCH *Kursor\_nazwa*** (polecenie zwróci wiersz w którym się znajduje kursor – bieżący wiersz)

Pobranie (określonego wiersza) z przechowaniem zmiennych

**FETCH *Kursor\_nazwa* INTO *lista\_zmiennych*** (polecenie zwróci wiersz w którym się znajduje kursor – bieżący wiersz ) lista zmiennych musi zawierać zmienną dla każdej kolumny instrukcji SELECT

**FETCH FIRST FROM *Kursor\_nazwa*** (Pobierz pierwszy wiersz)

**FETCH ABSOLUTE 5 FROM *Kursor\_nazwa*** (Pobierz piąty wiersz)

**FETCH NEXT** (Pobierz następny)

**FETCH PRIOR** (pobierz poprzedni)

**FETCH RELATIVE n** (zwraca wiersz oddalony o n pozycji względem bieżącego)

Do sprawdzenia, czy instrukcja FETCH zwróciła wiersz, służy zmienna systemowa  
**@@FETCH\_STATUS**

Przykład: WHILE @@FETCH\_STATUS = 0 . . . .

# Kursory - przykład

## PRZYKŁAD NR 1

--tworzenie kursora

**DECLARE** ProstyKursor **CURSOR**

**LOCAL**

**FOR SELECT** \* **FROM** Klienci

-- tworzenie zestawu kursora

**OPEN** ProstyKursor

-- Pobranie pierwszego wiersza

**FETCH** ProstyKursor

-- pobieraj kolejne wiersze kursora aż zmienna FETCH nie zwróci wiersza danych

**WHILE** @@FETCH\_STATUS = 0

**BEGIN**

**PRINT** 'POBRAŁEM'

**FETCH** ProstyKursor

**END**

-- Zwolnienie zestawu kursora

**CLOSE** ProstyKursor

-- Zwolnienie kursora

**DEALLOCATE** ProstyKursor

**GO**

# Kursory - przykład

-- **PRZYKŁAD NR 2.** (Pobranie kursora do zmiennych, tworzenie kursora)

```
DECLARE ProstyKursor CURSOR
```

```
    LOCAL
```

```
    keyset
```

```
    FOR SELECT tytuł,cena FROM Filmy
```

```
DECLARE @z_tytuł char(30), @z_cena int
```

```
-- tworzenie zestawu kursora
```

```
OPEN ProstyKursor
```

```
-- Pobranie pierwszej wiersza
```

```
FETCH first from ProstyKursor INTO @z_tytuł,@z_cena
```

```
-- wyświetlenie wyników
```

```
PRINT RTRIM(@z_tytuł) + ' jest w zmiennej'
```

```
PRINT RTRIM(@z_cena) + ' jest w zmiennej'
```

```
SELECT @z_tytuł as select1,@z_cena as select2
```

```
FETCH absolute 5 from ProstyKursor INTO @z_tytuł,@z_cena
```

```
PRINT RTRIM(@z_tytuł) + ' jest w zmiennej'
```

```
PRINT RTRIM(@z_cena) + ' jest w zmiennej'
```

```
SELECT @z_tytuł as select1,@z_cena as select2
```

```
-- Zwolnienie zestawu kursora
```

```
CLOSE ProstyKursor
```

```
-- Zwolnienie kursora
```

```
DEALLOCATE ProstyKursor
```

# Kursory – przykład: usuwanie powiązań między tabelami

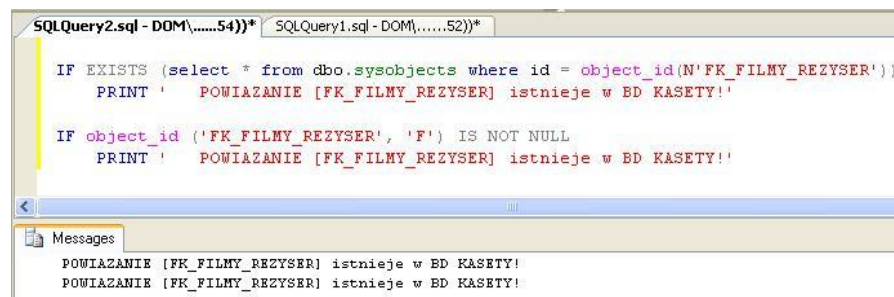
## PRZYKŁAD NR 3

```
DECLARE ProstyKursor CURSOR
LOCAL
FOR SELECT TABLE_NAME,CONSTRAINT_NAME FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE CONSTRAINT_TYPE = 'FOREIGN KEY'
-- deklaracja zmiennych
DECLARE @TABELA char(15),@POWIAZANIE varchar(30),@TEKST nvarchar(100)
OPEN ProstyKursor
-- Pobranie pierwszego wiersza do zmiennych
FETCH ProstyKursor INTO @TABELA,@POWIAZANIE
-- pobieraj kolejne wiersze kursora az zmienna FETCH nie zwróci wiersza danych
WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT 'POBRALEM: TABELA: '+@TABELA+' POWIAZANIE: '+@POWIAZANIE
        PRINT 'USUWAM POWIAZANIE: '+@POWIAZANIE
        SET @TEKST='ALTER TABLE '+@TABELA+' DROP CONSTRAINT ['+@POWIAZANIE+']'
        EXEC sp_executesql @TEKST
        -- ZLE: ALTER TABLE @TABELA DROP CONSTRAINT [@POWIAZANIE]
        FETCH ProstyKursor INTO @TABELA,@POWIAZANIE
    END
-- Zwolnienie zestawu kursora
CLOSE ProstyKursor
-- Zwolnienie kursora
DEALLOCATE ProstyKursor
GO

-- SELECT * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

# MS SQL Server

Dziękuję za uwagę!!!!!! .....  
Teraz ćwiczenie



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery2.sql - DOM\.....54))\*' and 'SQLQuery1.sql - DOM\.....52))\*'. The active tab displays a T-SQL query:

```
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'FK_FILMY_REZYSER'))  
    PRINT '    POWIAZANIE [FK_FILMY_REZYSER] istnieje w BD KASETY!'  
  
IF object_id ('FK_FILMY_REZYSER', 'F') IS NOT NULL  
    PRINT '    POWIAZANIE [FK_FILMY_REZYSER] istnieje w BD KASETY!'
```

Below the query editor, the 'Messages' pane shows the execution results:

```
POWIAZANIE [FK_FILMY_REZYSER] istnieje w BD KASETY!  
POWIAZANIE [FK_FILMY_REZYSER] istnieje w BD KASETY!
```

