

Bazy danych – Semestr 2

Zajęcia nr 6

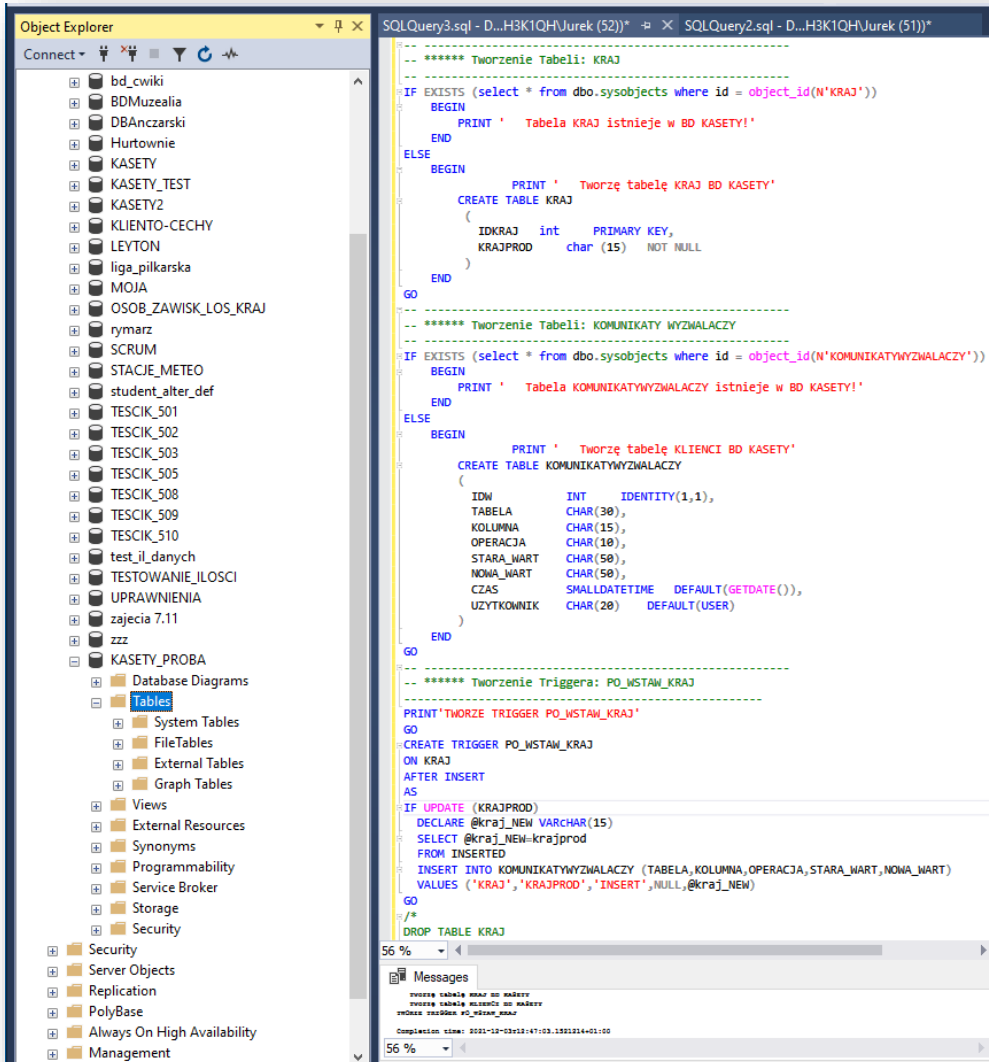
Pisanie skryptów
kontynuacja

Zakres zajęć

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

Procedury wyzwalane (ang.Triggers) – uwagi do poprzednich zajęć (1)

- Tworzenie triggera DODAJ_KRAJ w tabeli KRAJE (zapisuje rekord informacyjny w tabeli KOMUNIKATYWYZWALACZY)



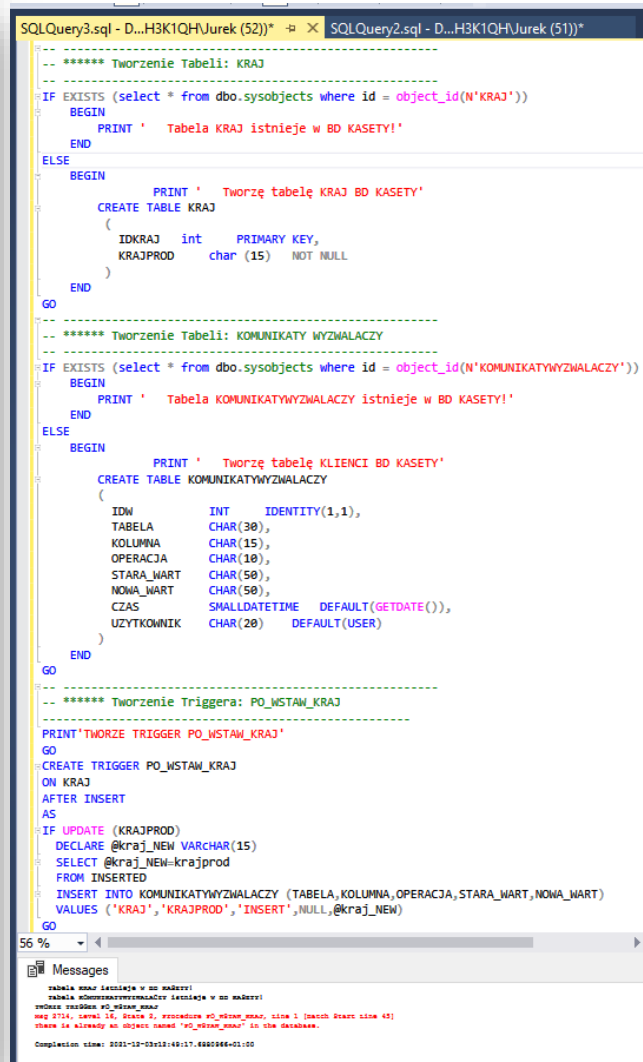
The screenshot shows the SQL Server Enterprise Manager on the left with a tree view of the database structure. The main window displays a SQL script for creating tables and a trigger. The script is as follows:

```
***** Tworzenie Tabeli: KRAJ
--
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KRAJ'))
BEGIN
    PRINT ' Tabela KRAJ istnieje w BD KASETY!'
END
ELSE
BEGIN
    PRINT ' Tworzę tabelę KRAJ BD KASETY'
    CREATE TABLE KRAJ
    (
        IDKRAJ int PRIMARY KEY,
        KRAJPROD char (15) NOT NULL
    )
END
GO

***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
--
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KOMUNIKATYWYZWALACZY'))
BEGIN
    PRINT ' Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!'
END
ELSE
BEGIN
    PRINT ' Tworzę tabelę KLIENCI BD KASETY'
    CREATE TABLE KOMUNIKATYWYZWALACZY
    (
        IDW INT IDENTITY(1,1),
        TABELA CHAR(30),
        KOLUMNA CHAR(15),
        OPERACJA CHAR(10),
        STARA_WART CHAR(50),
        NOWA_WART CHAR(50),
        CZAS SMALLDATETIME DEFAULT(GETDATE()),
        UZYTKOWNIK CHAR(20) DEFAULT(USER)
    )
END
GO

***** Tworzenie Triggera: PO_WSTAW_KRAJ
--
PRINT'TWORZE TRIGGER PO_WSTAW_KRAJ'
GO
CREATE TRIGGER PO_WSTAW_KRAJ
ON KRAJ
AFTER INSERT
AS
IF UPDATE (KRAJPROD)
DECLARE @kraj_NEW VARCHAR(15)
SELECT @kraj_NEW=krajprod
FROM INSERTED
INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)
GO
/*
DROP TABLE KRAJ

```



The screenshot shows the execution of the SQL script from the previous window. The Messages window at the bottom displays the following output:

```
***** Tworzenie Tabeli: KRAJ
--
Tabela KRAJ istnieje w BD KASETY!
***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
--
Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!
***** Tworzenie Triggera: PO_WSTAW_KRAJ
--
TWORZE TRIGGER PO_WSTAW_KRAJ
CREATE TRIGGER PO_WSTAW_KRAJ
ON KRAJ
AFTER INSERT
AS
IF UPDATE (KRAJPROD)
DECLARE @kraj_NEW VARCHAR(15)
SELECT @kraj_NEW=krajprod
FROM INSERTED
INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)

```

Procedury wyzwalane (ang.Triggers) – uwagi do poprzednich zajęć (2)

- Tworzenie triggera DODAJ_KRAJ w tabeli KRAJE (zapisuje rekord informacyjny w tabeli KOMUNIKATYWYZWALACZY)

The screenshot shows the SQL Server Enterprise Manager interface on the left, displaying a list of databases including 'KASETY_PROBA'. The main window displays a SQL script for creating the 'KRAJ' table and a trigger named 'PO_WSTAW_KRAJ'.

```
-- ***** Tworzenie Tabeli: KRAJ
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KRAJ'))
-- BEGIN
--     PRINT 'Tabela KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT 'Tworzę tabelę KRAJ BD KASETY'
--     CREATE TABLE KRAJ
--     (
--         IDKRAJ int PRIMARY KEY,
--         KRAJPROD char (15) NOT NULL
--     )
-- END
-- GO

-- ***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KOMUNIKATYWYZWALACZY'))
-- BEGIN
--     PRINT 'Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT 'Tworzę tabelę KLIENCI BD KASETY'
--     CREATE TABLE KOMUNIKATYWYZWALACZY
--     (
--         IDW int IDENTITY(1,1),
--         TABELA char(30),
--         KOLUMNA char(15),
--         OPERACJA char(10),
--         STARA_WART char(50),
--         NOWA_WART char(50),
--         CZAS smalldatetime DEFAULT(GETDATE()),
--         UZYTKOWNIK char(20) DEFAULT(USER)
--     )
-- END
-- GO

-- ***** Tworzenie Triggera: PO_WSTAW_KRAJ
-- IF OBJECT_ID ('PO_WSTAW_KRAJ','TR') IS NOT NULL
-- BEGIN
--     PRINT'USUWAM TRIGGER PO_WSTAW_KRAJ'
--     DROP TRIGGER PO_WSTAW_KRAJ
-- END
-- GO
-- PRINT'TWORZE TRIGGER PO_WSTAW_KRAJ'
-- GO
-- CREATE TRIGGER PO_WSTAW_KRAJ
-- ON KRAJ
-- AFTER INSERT
-- AS
-- IF UPDATE (KRAJPROD)
-- DECLARE @kraj_NEW VARCHAR(15)
-- SELECT @kraj_NEW=krajprod
-- FROM INSERTED
-- INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
-- VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)
-- GO
```

This screenshot is similar to the previous one, but it includes a 'Messages' window at the bottom right, which displays the execution results of the SQL script, including the creation of the 'KRAJ' table and the 'PO_WSTAW_KRAJ' trigger.

```
-- ***** Tworzenie Tabeli: KRAJ
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KRAJ'))
-- BEGIN
--     PRINT 'Tabela KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT 'Tworzę tabelę KRAJ BD KASETY'
--     CREATE TABLE KRAJ
--     (
--         IDKRAJ int PRIMARY KEY,
--         KRAJPROD char (15) NOT NULL
--     )
-- END
-- GO

-- ***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KOMUNIKATYWYZWALACZY'))
-- BEGIN
--     PRINT 'Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT 'Tworzę tabelę KLIENCI BD KASETY'
--     CREATE TABLE KOMUNIKATYWYZWALACZY
--     (
--         IDW int IDENTITY(1,1),
--         TABELA char(30),
--         KOLUMNA char(15),
--         OPERACJA char(10),
--         STARA_WART char(50),
--         NOWA_WART char(50),
--         CZAS smalldatetime DEFAULT(GETDATE()),
--         UZYTKOWNIK char(20) DEFAULT(USER)
--     )
-- END
-- GO

-- ***** Tworzenie Triggera: PO_WSTAW_KRAJ
-- IF OBJECT_ID ('PO_WSTAW_KRAJ','TR') IS NOT NULL
-- BEGIN
--     PRINT'USUWAM TRIGGER PO_WSTAW_KRAJ'
--     DROP TRIGGER PO_WSTAW_KRAJ
-- END
-- GO
-- PRINT'TWORZE TRIGGER PO_WSTAW_KRAJ'
-- GO
-- CREATE TRIGGER PO_WSTAW_KRAJ
-- ON KRAJ
-- AFTER INSERT
-- AS
-- IF UPDATE (KRAJPROD)
-- DECLARE @kraj_NEW VARCHAR(15)
-- SELECT @kraj_NEW=krajprod
-- FROM INSERTED
-- INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
-- VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)
-- GO
```

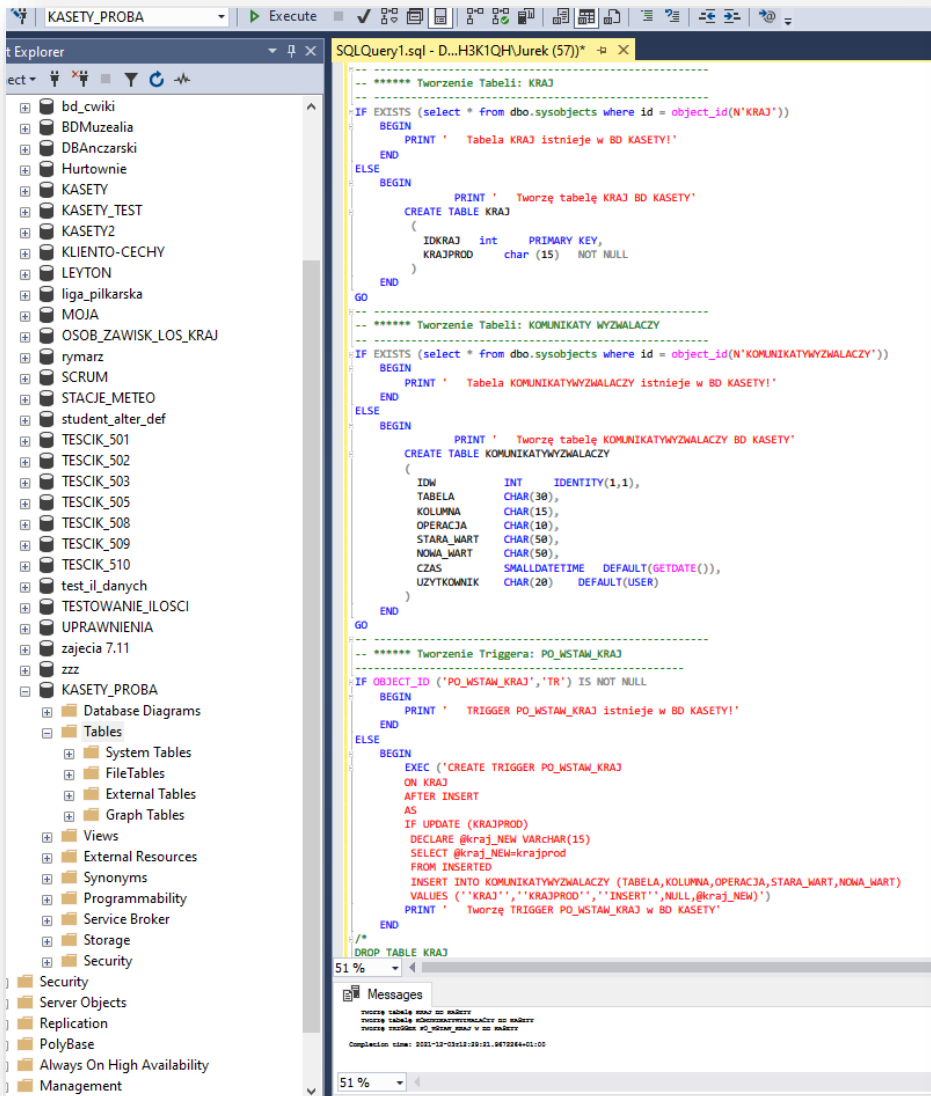
Messages

Tabela KRAJ została utworzona w BD KASETY!
Tabela KOMUNIKATYWYZWALACZY została utworzona w BD KASETY!
Trigger PO_WSTAW_KRAJ został utworzony w BD KASETY!

Compilation time: 2021-12-29 12:40:11.3218264-01:00

Procedury wyzwalane (ang.Triggers) – uwagi do poprzednich zajęć (3)

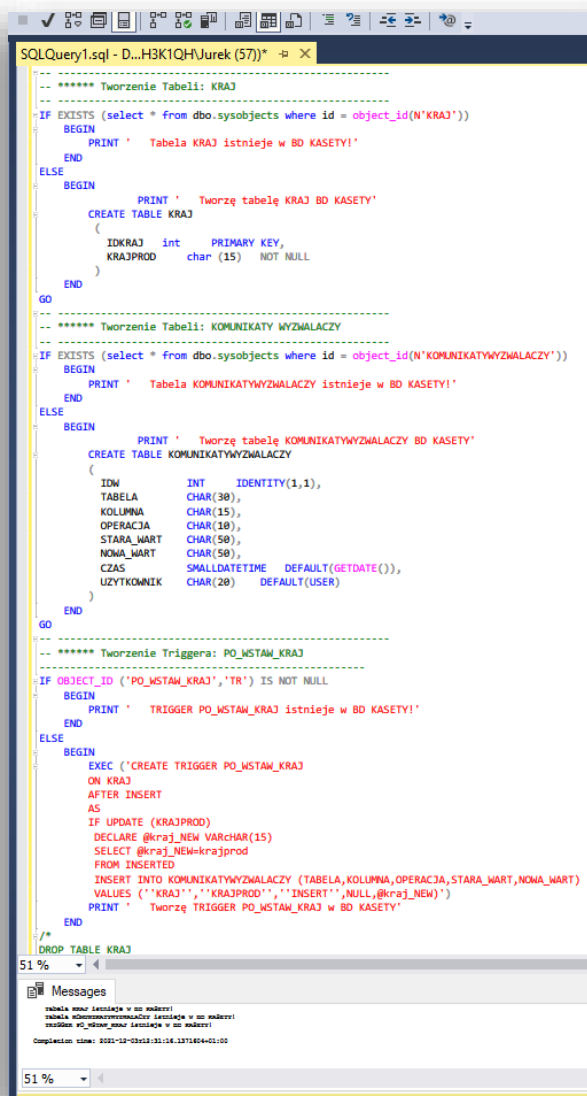
- Tworzenie triggera PO_WSTAW_KRAJ w tabeli KRAJE (zapisuje rekord informacyjny w tabeli KOMUNIKATYWYZWALACZY)



```
-- ***** Tworzenie Tabeli: KRAJ
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KRAJ'))
-- BEGIN
--     PRINT ' Tabela KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT ' Tworzę tabelę KRAJ BD KASETY!'
--     CREATE TABLE KRAJ
--     (
--         IDKRAJ int PRIMARY KEY,
--         KRAJPROD char (15) NOT NULL
--     )
-- END
-- GO

-- ***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KOMUNIKATYWYZWALACZY'))
-- BEGIN
--     PRINT ' Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT ' Tworzę tabelę KOMUNIKATYWYZWALACZY BD KASETY!'
--     CREATE TABLE KOMUNIKATYWYZWALACZY
--     (
--         IDW INT IDENTITY(1,1),
--         TABELA CHAR(30),
--         KOLUMNA CHAR(15),
--         OPERACJA CHAR(10),
--         STARA_WART CHAR(50),
--         NOWA_WART CHAR(50),
--         CZAS SMALLDATETIME DEFAULT(GETDATE()),
--         UZYTKOWNIK CHAR(20) DEFAULT(USER)
--     )
-- END
-- GO

-- ***** Tworzenie Triggera: PO_WSTAW_KRAJ
-- IF OBJECT_ID ('PO_WSTAW_KRAJ','TR') IS NOT NULL
-- BEGIN
--     PRINT ' TRIGGER PO_WSTAW_KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     EXEC ('CREATE TRIGGER PO_WSTAW_KRAJ
--     ON KRAJ
--     AFTER INSERT
--     AS
--     IF UPDATE (KRAJPROD)
--     DECLARE @kraj_NEW VARCHAR(15)
--     SELECT @kraj_NEW=krajprod
--     FROM INSERTED
--     INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
--     VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)')
--     PRINT ' Tworzę TRIGGER PO_WSTAW_KRAJ w BD KASETY'
-- END
-- GO
-- DROP TABLE KRAJ
```



```
-- ***** Tworzenie Tabeli: KRAJ
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KRAJ'))
-- BEGIN
--     PRINT ' Tabela KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT ' Tworzę tabelę KRAJ BD KASETY!'
--     CREATE TABLE KRAJ
--     (
--         IDKRAJ int PRIMARY KEY,
--         KRAJPROD char (15) NOT NULL
--     )
-- END
-- GO

-- ***** Tworzenie Tabeli: KOMUNIKATY WYZWALACZY
-- IF EXISTS (select * from dbo.sysobjects where id = object_id(N'KOMUNIKATYWYZWALACZY'))
-- BEGIN
--     PRINT ' Tabela KOMUNIKATYWYZWALACZY istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     PRINT ' Tworzę tabelę KOMUNIKATYWYZWALACZY BD KASETY!'
--     CREATE TABLE KOMUNIKATYWYZWALACZY
--     (
--         IDW INT IDENTITY(1,1),
--         TABELA CHAR(30),
--         KOLUMNA CHAR(15),
--         OPERACJA CHAR(10),
--         STARA_WART CHAR(50),
--         NOWA_WART CHAR(50),
--         CZAS SMALLDATETIME DEFAULT(GETDATE()),
--         UZYTKOWNIK CHAR(20) DEFAULT(USER)
--     )
-- END
-- GO

-- ***** Tworzenie Triggera: PO_WSTAW_KRAJ
-- IF OBJECT_ID ('PO_WSTAW_KRAJ','TR') IS NOT NULL
-- BEGIN
--     PRINT ' TRIGGER PO_WSTAW_KRAJ istnieje w BD KASETY!'
-- END
-- ELSE
-- BEGIN
--     EXEC ('CREATE TRIGGER PO_WSTAW_KRAJ
--     ON KRAJ
--     AFTER INSERT
--     AS
--     IF UPDATE (KRAJPROD)
--     DECLARE @kraj_NEW VARCHAR(15)
--     SELECT @kraj_NEW=krajprod
--     FROM INSERTED
--     INSERT INTO KOMUNIKATYWYZWALACZY (TABELA,KOLUMNA,OPERACJA,STARA_WART,NOWA_WART)
--     VALUES ('KRAJ','KRAJPROD','INSERT',NULL,@kraj_NEW)')
--     PRINT ' Tworzę TRIGGER PO_WSTAW_KRAJ w BD KASETY'
-- END
-- GO
-- DROP TABLE KRAJ
```

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 jest zabroniona',16,1)
```

Poziom błędu, stan błędu

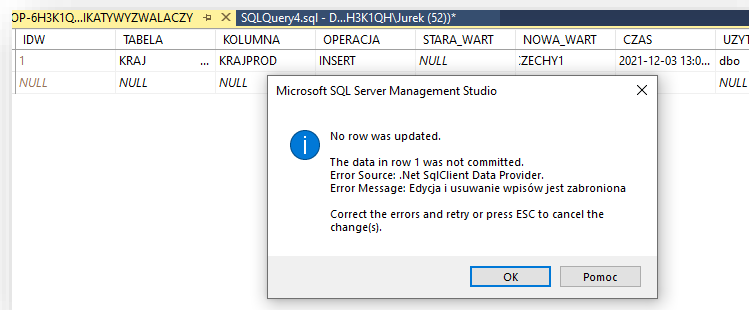
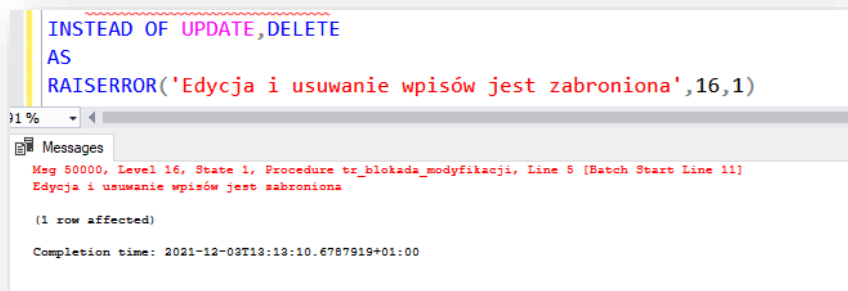
Po utworzeniu triggera usuńmy/zmodyfikujmy wiersz danych w tabeli

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 jest zabroniona',10
```

Po utworzeniu triggera usuńmy/zmodyfikujmy wiersz danych w tabeli

KomunikatyWyzwalarczy

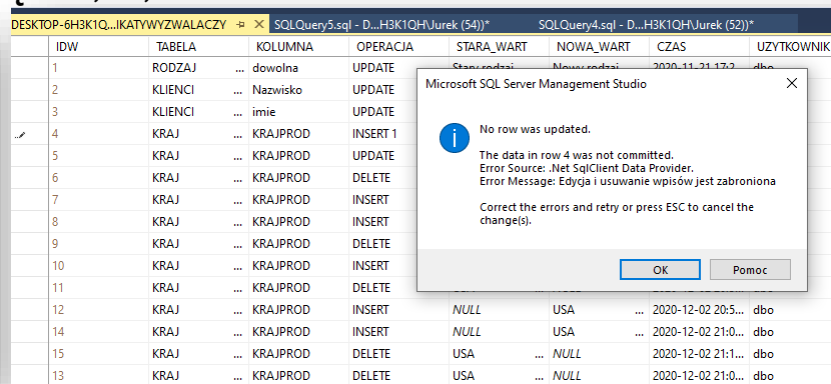
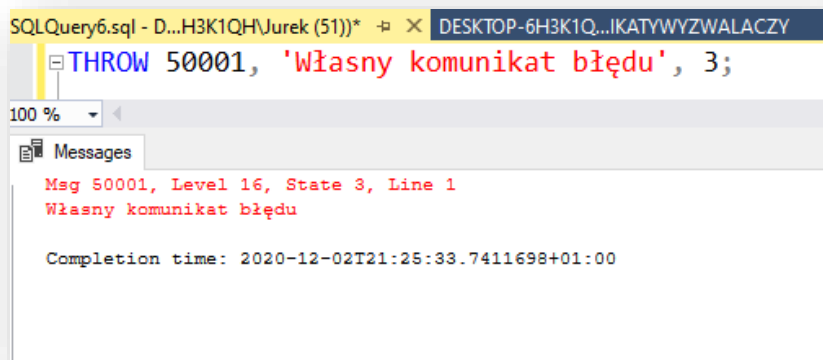
.....udało się ? Nie.. Sukces !!!!!!!!!!!!!!!!!!!!!

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:

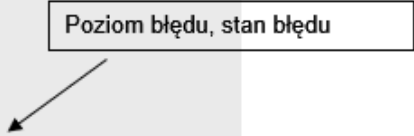
THROW 50001, 'Własny komunikat błędu', 3;



Procedury wyzwalane (ang.Triggers) - 3

Składnia triggera

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



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

W **RAISERROR** przechwytujemy wartość ważności, natomiast w **THROW** ustawiony jest na sztywno i zawsze zwraca wartość równą 16. W **RAISERROR** błąd zwrócony z ID > 50000 w przypadku braku definicji w tabeli systemowej sys

```
INSTEAD OF UPDATE,DELETE
AS
RAISERROR('Edycja i usuwanie wpisów jest zabroniona',16,1)
```

Messages

Msg 50000, Level 16, State 1, Procedure tr_blokada_modyfikacji, Line 5 [Batch Start Line 11]
Edycja i usuwanie wpisów jest zabroniona

(1 row affected)

Completion time: 2021-12-03T13:13:10.6787919+01:00

SQLQuery6.sql - D...H3K1QH\Jurek (51)) * X DESKTOP-6H3K1Q...IKATYWYZWALACZY

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

100 %

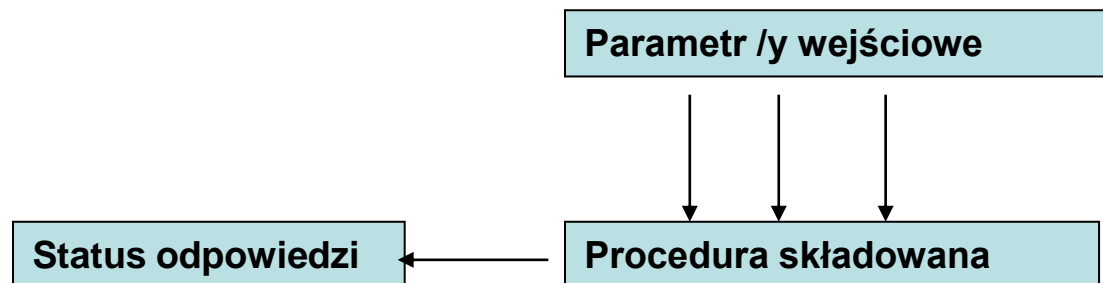
Messages

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

Completion time: 2020-12-02T21:25:33.7411698+01:00

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 (1)

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 (1)

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.

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>)

Procedury składowane (2)

- **TRY/CATCH** to konstrukcja, która występuje w wielu językach programowania. **TRY/CATCH** pomaga odseparować daną logikę zapytania która w szczególnych przypadkach może zwrócić błąd.
- W bloku **TRY** wykonujemy zapytanie, transakcję <np.. insert, update, delete),.
- W bloku **CATCH** - w przypadku wystąpienia błędu wychytujemy ten błąd i obsługujemy.

Procedury składowane (2)

Przykład

2. Procedura dodająca nowy kraj do bazy danych

```
CREATE PROCEDURE ZAPISZ_NOWY_KRAJ1
```

```
(@idkraj int,  
@krajprod char(15),  
@Komunikat varchar(200) output)  
AS
```

Parametry wejściowe

Parametr wyjściowy

```
BEGIN TRY
```

```
INSERT INTO kraj(idkraj,krajprod) VALUES (@idkraj,@krajprod)
```

```
set @komunikat = 'Tworze nowy kraj - pomyślnie zapisany'
```

Wykonanie instrukcji

```
END TRY
```

```
BEGIN CATCH
```

```
set @komunikat = 'błąd tworzenia nowego kraju- BRAK ZAPISU'
```

```
SELECT ErrorNumber = ERROR_NUMBER(),  
ErrorSeverity = ERROR_SEVERITY(),  
ErrorState = ERROR_STATE(),  
ErrorProcedure = ERROR_PROCEDURE(),  
ErrorLine = ERROR_LINE(),  
ErrorMessage = ERROR_MESSAGE()
```

Obsługa błędu

```
END CATCH
```

Procedury składowane (2)

OBSŁUGA BŁĘDÓW

- **ERROR_LINE** - numer linii (w ciele kodu) w której pojawił się błąd;
- **ERROR_NUMBER** (message id) - identyfikator błędu z tabeli systemowej sys.messages. Podobne do parametru @@**ERROR** z tym, że zwraca ten sam numer przez cały czas trwania bloku **CATCH**;
(id≤50000 – błędy systemowe,
ID>50000 błędy zdefiniowane przez użytkownika)
- **ERROR_SEVERITY** - rygor zgłoszonego błędu (ważność błędu <0-25>. Blok **CATCH** zostaje użyty jeżeli zgłoszony błąd ma numer 11 lub większy.
 - Poziom błędów od 11 do 16 oznacza zwykle błędy użytkownika (programisty) lub kodu.
 - Poziom 17 do 25 to zwykle błędy oprogramowania lub sprzętu w przypadku których dalsze wykonywanie instrukcji nie jest możliwe;
- **ERROR_STATE** - czasami używane przez system do zwrócenia większej liczby informacji o błędzie.
- **ERROR_MESSAGE** - pełny tekst komunikatu o błędzie zawierający wszystkie parametry, np. nazwa obiektu;

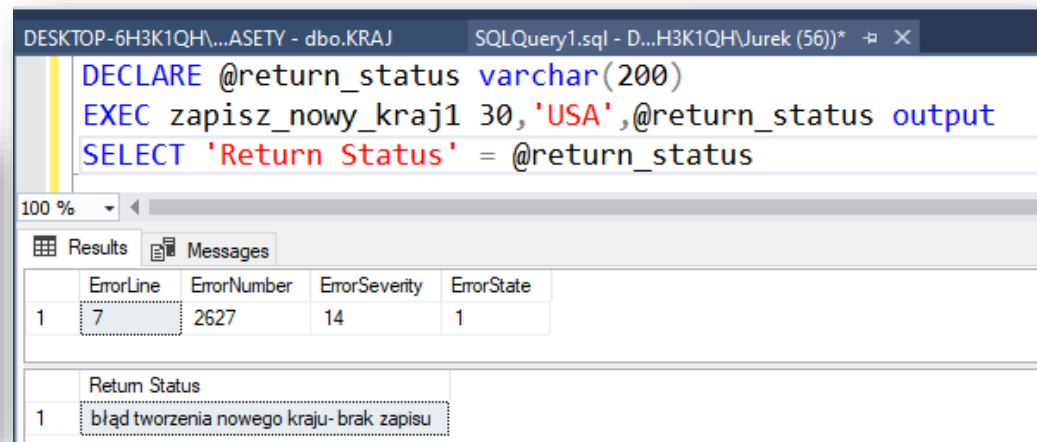
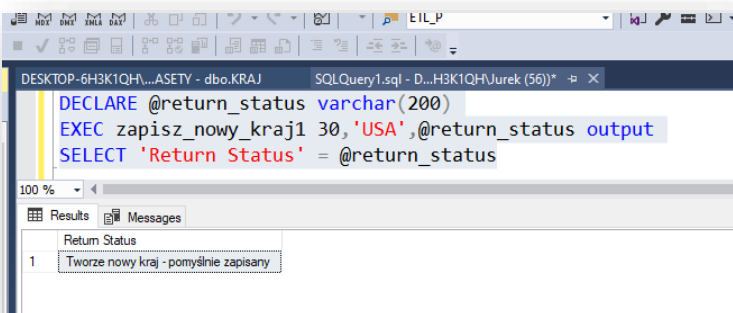
Procedury składowane (2)

Przykład wywołania

Kontrola poprawności wykonania.

```
DECLARE @return_status varchar(200)
EXEC zapisz_nowy_kraj1 30 'USA',@return_status output
SELECT 'Return Status' = @return_status
GO
```

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

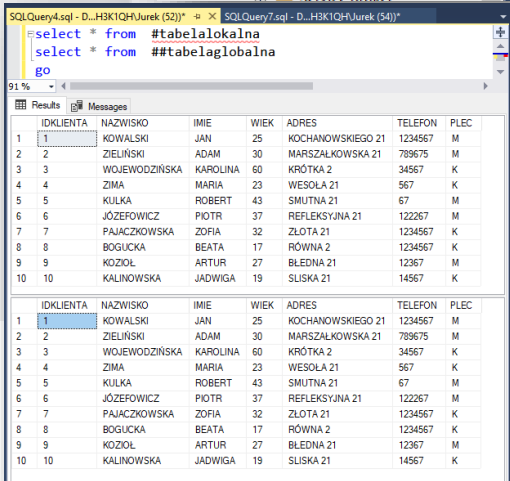
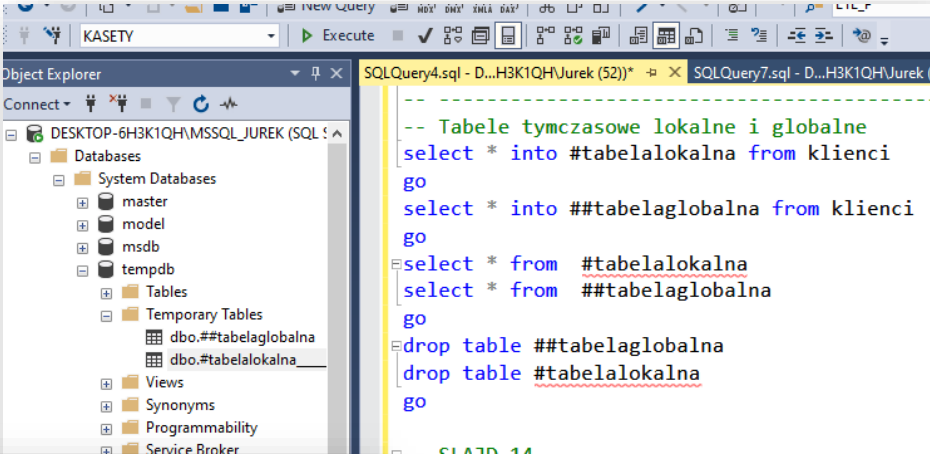


Tabele tymczasowe lokalne i globalne

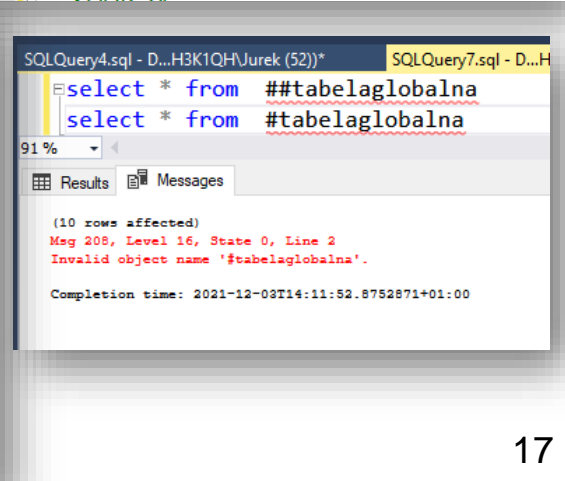
- Tabele tymczasowe znajdują się: przechodząc w drzewie obiektów SSMS. Wybierając Databases > System databases > tempdb > Temporary Tables.
- W **SQL Server** mamy do wyboru 2 rodzaje **tabel tymczasowych**:
 - ✓ lokalne
 - ✓ globalne.
- Tabele tymczasowe lokalne w **SQL Server** działają w obrębie danej sesji.

Przykład:

```
select * into #tabelalokalna from klienci
go
select * into ##tabelaglobalna from klienci
go
select * from #tabelalokalna
select * from ##tabelaglobalna
go
drop table ##tabelaglobalna
drop table #tabelalokalna
go
```



IDKlienta	Nazwisko	Imie	Wiek	Adres	Telefon	Plec
1	KOWALSKI	JAN	25	KOCHANOWSKIEGO 21	1234567	M
2	ZIELIŃSKI	ADAM	30	MARSZAŁKOWSKA 21	789675	M
3	WOJEWODZIŃSKA	KAROLINA	60	KRÓTKA 2	34567	K
4	ZIMA	MARIA	23	WESOŁA 21	567	K
5	KULKA	ROBERT	43	SMUTNA 21	67	M
6	JÓZEFOWICZ	PIOTR	37	REFLEKSYJNA 21	122267	M
7	PAJĄCZKOWSKA	ZOFIA	32	ZŁOTA 21	1234567	K
8	BOGUĆKA	BEATA	17	RÓWNA 2	1234567	K
9	KOZIOL	ARTUR	27	BIEDNA 21	12367	M
10	KALINOWSKA	JADWIGA	19	SLISKA 21	14567	K



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

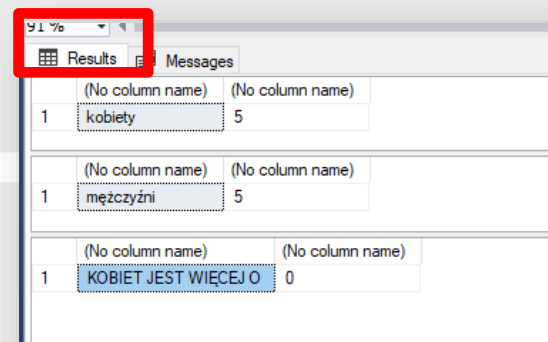
Przykład

Napisać funkcję pokazującą informację w przypadku gdy:

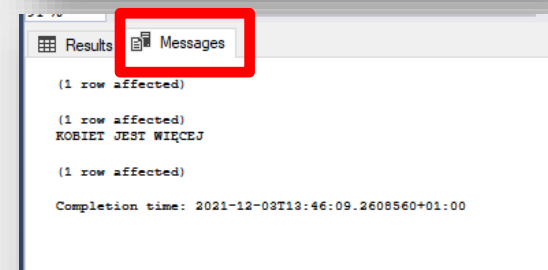
- mężczyźni jest więcej od kobiet (w tabeli klienci) wypisany będzie komunikat „MĘŻCZYŹN JEST WIĘCEJ O” i podana liczba
- kobiet jest więcej od mężczyzn (w tabeli klienci) wypisany będzie komunikat „Kobiet jest więcej o” i podana liczba

```
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
```



	(No column name)	(No column name)
1	kobiety	5
1	mężczyźni	5
1	Kobiet jest więcej O	0



(1 row affected)

(1 row affected)

Kobiet jest więcej

(1 row affected)

Completion time: 2021-12-03T13:46:09.2608560+01:00

Instrukcja warunkowa IF (2) – bez wyniku różnicy

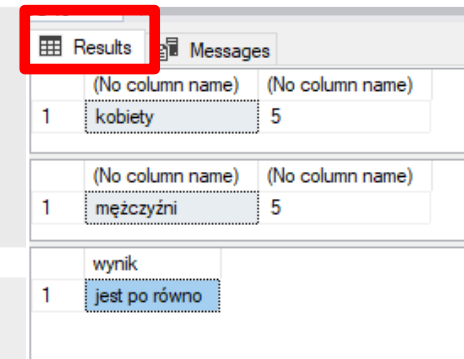
Przykład

Napisać funkcję pokazującą informację w przypadku gdy:

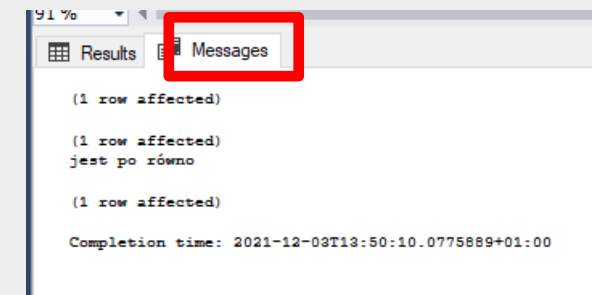
- mężczyźni jest więcej od kobiet (w tabeli klienci) wypisany będzie komunikat „MĘŻCZYŹN JEST WIĘCEJ O” i podana liczba
- kobiet jest więcej od mężczyzn (w tabeli klienci) wypisany będzie komunikat „Kobiet jest więcej O” i podana liczba

```
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
```

```
PRINT IIF (@m > @k, 'mężczyzn jest więcej',
  IIF (@m < @k, 'kobiet jest więcej',
    'jest po równo'))
SELECT IIF (@m > @k, 'mężczyzn jest więcej',
  IIF (@m < @k, 'kobiet jest więcej',
    'jest po równo')) as wynik
```



	(No column name)	(No column name)
1	kobiety	5
	(No column name)	(No column name)
1	mężczyźni	5
wynik		
1	jest po równo	



(1 row affected)

(1 row affected)

jest po równo

(1 row affected)

Completion time: 2021-12-03T13:50:10.0775889+01:00

Instrukcja warunkowa IF (2) – z wynikiem różnicy

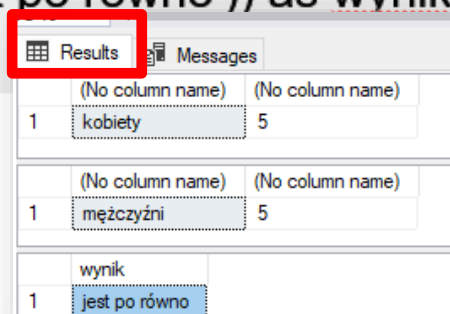
Przykład

Napisać funkcję pokazującą informację w przypadku gdy:

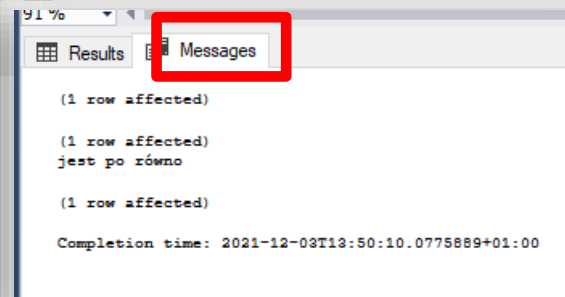
- mężczyźni jest więcej od kobiet (w tabeli klienci) wypisany będzie komunikat „MĘŻCZYŹN JEST WIĘCEJ O” i podana liczba
- kobiet jest więcej od mężczyzn (w tabeli klienci) wypisany będzie komunikat „Kobiet jest więcej o” i podana liczba

```
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
```

```
PRINT IIF (@m > @k, concat('mężczyzn jest więcej o: ',RTRIM(@m-@k)),
IIF (@m < @k, CONCAT('mężczyzn jest mniej o: ',RTRIM(@k-@m)),
'jest po równo'))
SELECT IIF (@m > @k, concat('mężczyzn jest więcej o: ',RTRIM(@m-@k)),
IIF (@m < @k, CONCAT('mężczyzn jest mniej o: ',
RTRIM(@k-@m)), 'jest po równo')) as wynik z concat
```



	(No column name)	(No column name)
1	kobiety	5
	(No column name)	(No column name)
1	mężczyźni	5



(1 row affected)

(1 row affected)
jest po równo

(1 row affected)

Completion time: 2021-12-03T13:50:10.0775889+01:00

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

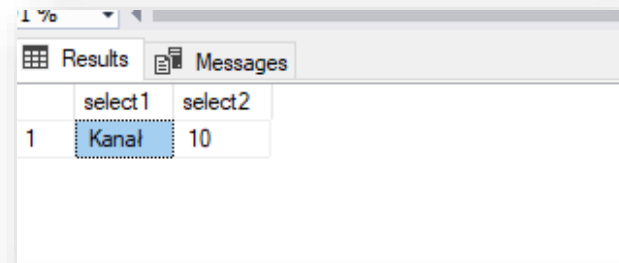
```
DEALLOCATE ProstyKursor
```

```
GO
```

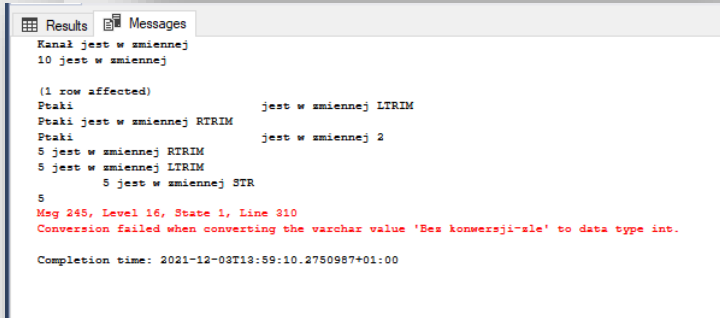
Kursory - przykład

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

```
DECLARE ProstýKursor CURSOR
    LOCAL
    keyset
    FOR SELECT tytuł,cena FROM Filmy
DECLARE @z_tytuł char(30), @z_cena int
-- tworzenie zestawu kursora
OPEN ProstýKursor
-- Pobranie pierwszego wiersza
FETCH first from ProstýKursor 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 ProstýKursor 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
PRINT @z_cena
PRINT @z_cena + 'Bez konwersji-zle'
-- Zwolnienie zestawu kursora
CLOSE ProstýKursor
-- Zwolnienie kursora
DEALLOCATE ProstýKursor
```



	select1	select2
1	Kanał	10



```
(1 row affected)
Ptaki jest w zmiennej LTRIM
Ptaki jest w zmiennej RTRIM
Ptaki jest w zmiennej 2
5 jest w zmiennej RTRIM
5 jest w zmiennej LTRIM
5 jest w zmiennej STR
5
Msg 245, Level 16, State 1, Line 310
Conversion failed when converting the varchar value 'Bez konwersji-zle' to data type int.
Completion time: 2021-12-08T13:59:10.2750987+01:00
```

Kursory – przykład: usuwanie powiązań między tabelami

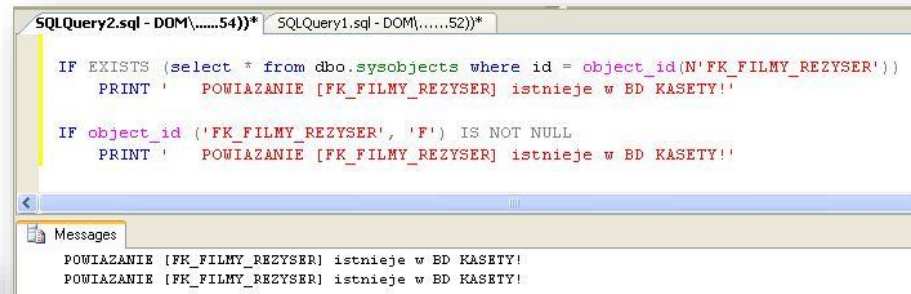
PRZYKŁAD NR 3

```
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, the 'Messages' pane shows two identical messages:

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

