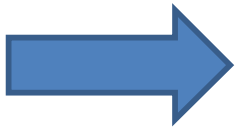
Procedury składowane

Definicja

- **Procedura składowana** to fragment kodu SQL składający się z instrukcji deklaratywnych i/lub proceduralnych i przechowywany w katalogu RDBMS
- Musi być wywoływana jawnie

Tworzenie procedury

```
SELECT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    product_name;
```



```
CREATE PROCEDURE uspProductList
AS
BEGIN
    SELECT
        product_name,
        list_price
    FROM
        production.products
    ORDER BY
        product_name;
END;
```

Uruchamianie procedury

- `EXECUTE sp_name;`
- `EXEC sp_name;`
  - `EXEC uspProductList`
- **Programmability > Stored Procedures**
- **select * from sys.procedures**

Modyfikowanie procedury

```
ALTER PROCEDURE uspProductList
    AS
    BEGIN
        SELECT
            product_name,
            list_price
        FROM
            production.products
        ORDER BY
            list_price
    END;
EXEC uspProductList;
```

Usuwanie procedury

```
DROP PROCEDURE sp_name;

DROP PROC sp_name;


DROP PROCEDURE uspProductList;
```

Procedura z 1 parametrem

```
CREATE PROCEDURE
uspFindProducts
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   ORDER BY
      list_price;
END;
```

```
ALTER PROCEDURE uspFindProducts(@min_list_price AS DECIMAL)
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      list_price >= @min_list_price
   ORDER BY
      list_price;
END;
EXEC uspFindProducts 100;
```

Procedura z kilkoma parametrami

```
ALTER PROCEDURE uspFindProducts(
   @min_list_price AS DECIMAL
   ,@max_list_price AS DECIMAL
)
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      list_price >= @min_list_price AND
      list_price <= @max_list_price
   ORDER BY
      list_price;
END;

EXECUTE uspFindProducts 900, 1000;
```

Procedura z kilkoma parametrami

```
EXECUTE uspFindProducts
    @min_list_price = 900,
    @max_list_price = 1000;
```

Parametry tekstowe

```
ALTER PROCEDURE uspFindProducts(
   @min_list_price AS DECIMAL
   ,@max_list_price AS DECIMAL
   ,@name AS VARCHAR(max)
)
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      list_price >= @min_list_price AND
      list_price <= @max_list_price AND
      product_name LIKE '%' + @name + '%'
   ORDER BY
      list_price;
END;

EXECUTE uspFindProducts
   @min_list_price = 900,
   @max_list_price = 1000,
   @name = 'Trek';
```

Parametry opcjonalne

```
ALTER PROCEDURE uspFindProducts(
   @min_list_price AS DECIMAL = 0
   ,@max_list_price AS DECIMAL = 999999
   ,@name AS VARCHAR(max)
)
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      list_price >= @min_list_price AND
      list_price <= @max_list_price AND
      product_name LIKE '%' + @name + '%'
   ORDER BY
      list_price;
END;

EXECUTE uspFindProducts
   @name = 'Trek’;
EXECUTE uspFindProducts
   @min_list_price = 6000,
   @name = 'Trek';
```

NULL jako wartość deafultowa

```
ALTER PROCEDURE uspFindProducts(
   @min_list_price AS DECIMAL = 0
   ,@max_list_price AS DECIMAL = NULL
   ,@name AS VARCHAR(max)
)
AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      list_price >= @min_list_price AND
      (@max_list_price IS NULL OR list_price <= @max_list_price) AND
      product_name LIKE '%' + @name + '%'
   ORDER BY
      list_price;
END;

EXECUTE uspFindProducts
   @min_list_price = 500,
   @name = 'Haro';
```

Zmienna - deklaracja

- Jako licznik pętli - ile razy wykonywana jest pętla
- Przechowywanie wartości, która ma być testowana przez instrukcję taką jak WHILE
- Aby przechowywać wartość zwróconą przez procedurę składowaną lub funkcję

```
DECLARE @model_year (AS) SMALLINT;
DECLARE @model_year SMALLINT,
        @product_name VARCHAR(MAX);
SET @model_year = 2018;
```

Zmienna – użycie w zapytaniu

```
DECLARE @model_year SMALLINT;

SET @model_year = 2018;

SELECT
   product_name,
   model_year,
   list_price
FROM
   production.products
WHERE
   model_year = @model_year
ORDER BY
   product_name;
```

Przechowywanie wyniku zapytania w zmiennej

```
DECLARE @product_count INT;
SET @product_count = (
   SELECT
      COUNT(*)
   FROM
      production.products
);
SELECT @product_count;
PRINT @product_count;
PRINT 'The number of products is ' +
CAST(@product_count AS VARCHAR(MAX));
```

Wybór rekordu do zmiennych

```
DECLARE
   @product_name VARCHAR(MAX),
   @list_price DECIMAL(10,2);
SELECT
   @product_name = product_name,
   @list_price = list_price
FROM
   production.products
WHERE
   product_id = 100;

SELECT
   @product_name AS product_name,
   @list_price AS list_price;
```

Kumulowanie wartości w zmienną

```
CREATE  PROC uspGetProductList(
   @model_year SMALLINT
) AS
BEGIN
   DECLARE @product_list VARCHAR(MAX);

   SET @product_list = '';

   SELECT
      @product_list = @product_list + product_name
                  + CHAR(10)
   FROM
      production.products
   WHERE
      model_year = @model_year
   ORDER BY
      product_name;

   PRINT @product_list;
END;

EXEC uspGetProductList 2018
```

Tworzenie parametrów wyjściowych

parameter_name data_type OUTPUT

```
CREATE PROCEDURE uspFindProductByModel (
   @model_year SMALLINT,
   @product_count INT OUTPUT
) AS
BEGIN
   SELECT
      product_name,
      list_price
   FROM
      production.products
   WHERE
      model_year = @model_year;

   SELECT @product_count = @@ROWCOUNT;
END;
```

Wywoływanie procedur składowanych z parametrami wyjściowymi

```
DECLARE @count INT;
EXEC uspFindProductByModel
    @model_year = 2018,
    @product_count = @count OUTPUT;

(EXEC uspFindProductByModel 2018, @
OUTPUT;)

SELECT @count AS 'Number of products found';
```

Results | Messages

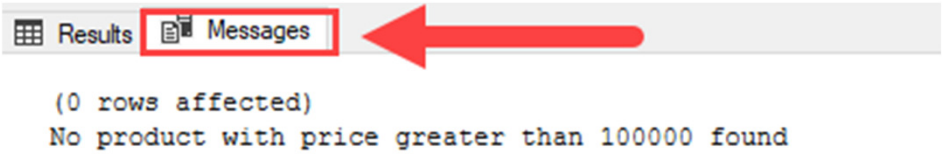
	product_name	list_price
1	Trek 820 - 2018	379.99
2	Trek Marlin 5 - 2018	489.99
3	Trek Marlin 6 - 2018	579.99
4	Trek Fuel EX 8 29 - 2018	3199.99
5	Trek Marlin 7 - 2017/2018	749.99
6	Trek Ticket S Frame - 2018	1469.99
7	Trek X-Caliber 8 - 2018	999.99
8	Trek Kids' Neko - 2018	469.99
9	Trek Fuel EX 7 29 - 2018	2499.99
10	Surly Krampus Frameset - 2018	2499.99

	Number of products found
1	204

Instrukcja BEGIN…END

```
BEGIN
   { sql_statement | statement_block}
END
```

```
BEGIN
   SELECT
      product_id,
      product_name
   FROM
      production.products
   WHERE
      list_price > 100000;

   IF @@ROWCOUNT = 0
      PRINT 'No product with price greater than 100000 found';
END
```



Zagnieżdżanie BEGIN…END

```
BEGIN
   DECLARE @name VARCHAR(MAX);

   SELECT TOP 1
      @name = product_name
   FROM
      production.products
   ORDER BY
      list_price DESC;

   IF @@ROWCOUNT <> 0
   BEGIN
      PRINT 'The most expensive product is ' + @name
   END
   ELSE
   BEGIN
      PRINT 'No product found';
   END;
END
```

Instrukcja IF

```
IF boolean_expression
BEGIN
   { statement_block }
END
BEGIN
   DECLARE @sales INT;

   SELECT
      @sales = SUM(list_price * quantity)
   FROM
      sales.order_items i
      INNER JOIN sales.orders o ON o.order_id = i.order_id
   WHERE
      YEAR(order_date) = 2018;

   SELECT @sales;

   IF @sales > 1000000
   BEGIN
      PRINT 'Great! The sales amount in 2018 is greater than 1,000,000';
   END
END
```

Instrukcja IF..ELSE

```
IF Boolean_expression
BEGIN
   -- Statement block executes when the Boolean
expression is TRUE
END
ELSE
BEGIN
   -- Statement block executes when the Boolean
expression is FALSE
END
```

Instrukcja IF..ELSE

```
BEGIN
   DECLARE @sales INT;

   SELECT
      @sales = SUM(list_price * quantity)
   FROM
      sales.order_items i
      INNER JOIN sales.orders o ON o.order_id = i.order_id
   WHERE
      YEAR(order_date) = 2017;

   SELECT @sales;

   IF @sales > 10000000
   BEGIN
      PRINT 'Great! The sales amount in 2018 is greater than 10,000,000';
   END
   ELSE
   BEGIN
      PRINT 'Sales amount in 2017 did not reach 10,000,000';
   END
END
```

Zagnieżdżanie IF..ELSE

```
BEGIN
   DECLARE @x INT = 10,
           @y INT = 20;

   IF (@x > 0)
   BEGIN
      IF (@x < @y)
         PRINT 'x > 0 and x < y';
      ELSE
         PRINT 'x > 0 and x >= y';
   END
END
```

Instrukcja WHILE

WHILE Boolean_expression

{ sql_statement | statement_block}

```
DECLARE @counter INT = 1;
WHILE @counter <= 5
BEGIN
   PRINT @counter;
   SET @counter = @counter + 1;
END
```

Instrukcja BREAK

```
WHILE Boolean_expression
BEGIN
   -- statements
  IF condition
      BREAK;
   -- other statements
END

WHILE Boolean_expression1
BEGIN
   -- statement
   WHILE Boolean_expression2
   BEGIN
      IF condition
         BREAK;
   END
END
```

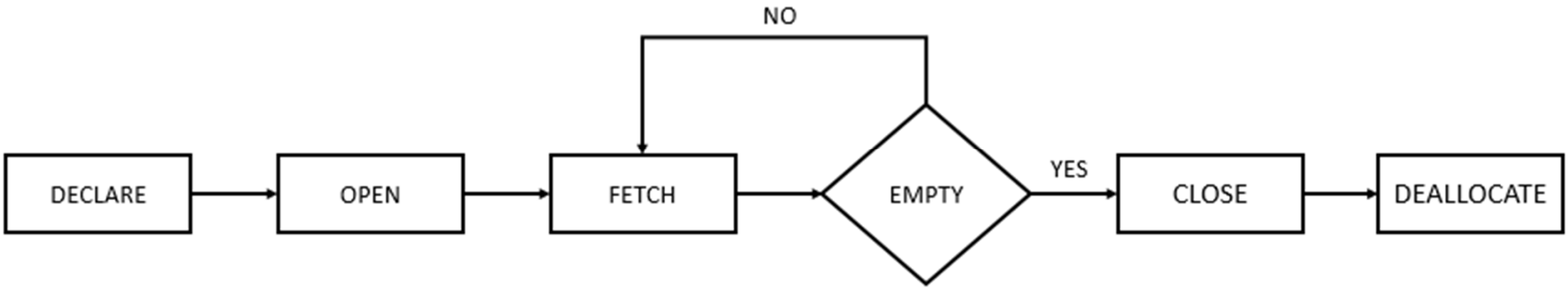
Instrukcja BREAK

```
DECLARE @counter INT = 0;

WHILE @counter <= 5
BEGIN
   SET @counter = @counter + 1;
   IF @counter = 4
      BREAK;
   PRINT @counter;
END
```

Instrukcja CONTINUE

```
WHILE Boolean_expression
BEGIN
   -- code to be executed
   IF condition
      CONTINUE;
   -- code will be skipped if the condition is met
END

DECLARE @counter INT = 0;
WHILE @counter < 5
BEGIN
   SET @counter = @counter + 1;
   IF @counter = 3
      CONTINUE;
   PRINT @counter;
END
```

Kursor



Kursor

1. Deklarowanie

```
DECLARE cursor_name CURSOR FOR select_statement;
```

2. Otwieranie

```
OPEN cursor_name;
```

3. Pobranie wiersz z kursora do jednej lub więcej zmiennych

```
FETCH NEXT FROM cursor INTO variable_list;
WHILE @@FETCH_STATUS = 0
  BEGIN
    FETCH NEXT FROM cursor_name;
  END;
```

4. Zamknięcie kursora

```
CLOSE cursor_name;
```

5. Zwolnienie kursora

```
DEALLOCATE cursor_name;
```

Kursor

```
DECLARE
   @product_name VARCHAR(MAX),
   @list_price   DECIMAL;

DECLARE cursor_product CURSOR
FOR SELECT
      product_name,
      list_price
   FROM
      production.products;

OPEN cursor_product;

FETCH NEXT FROM cursor_product INTO
   @product_name,
   @list_price;

WHILE @@FETCH_STATUS = 0
   BEGIN
      PRINT @product_name + CAST(@list_price AS varchar);
      FETCH NEXT FROM cursor_product INTO
         @product_name,
         @list_price;
   END;

CLOSE cursor_product;

DEALLOCATE cursor_product;
```

production.products

* product_id
product_name
brand_id
category_id
model_year
list_price

TRY CATCH

```
BEGIN TRY
   -- statements that may cause exceptions
END TRY
BEGIN CATCH
   -- statements that handle exception
END CATCH
```

Funkcje bloku CATCH

- ERROR_LINE() zwraca numer wiersza, w którym wystąpił wyjątek.
- ERROR_MESSAGE() zwraca pełny tekst wygenerowanego komunikatu o błędzie.
- ERROR_PROCEDURE() zwraca nazwę procedury składowanej lub triggera, w którym wystąpił błąd.
- ERROR_NUMBER() zwraca numer błędu, który wystąpił.
- ERROR_SEVERITY() zwraca poziom istotności błędu, który wystąpił.
- ERROR_STATE() zwraca numer stanu błędu, który wystąpił.

Zagnieżdżanie TRY CATCH

```
BEGIN TRY
   --- statements that may cause exceptions
END TRY
BEGIN CATCH
   -- statements to handle exception
   BEGIN TRY
      --- nested TRY block
   END TRY
   BEGIN CATCH
      --- nested CATCH block
   END CATCH
END CATCH
```

TRY CATCH

```
CREATE PROC usp_divide(
   @a decimal,
   @b decimal,
   @c decimal output
) AS
BEGIN
   BEGIN TRY
      SET @c = @a / @b;
   END TRY
   BEGIN CATCH
      SELECT
         ERROR_NUMBER() AS ErrorNumber
         ,ERROR_SEVERITY() AS ErrorSeverity
         ,ERROR_STATE() AS ErrorState
         ,ERROR_PROCEDURE() AS ErrorProcedure
         ,ERROR_LINE() AS ErrorLine
         ,ERROR_MESSAGE() AS ErrorMessage;
   END CATCH
END;
DECLARE @r decimal;
EXEC usp_divide 10, 2, @r output;
PRINT @r;
DECLARE @r2 decimal;
EXEC usp_divide 10, 0, @r2 output;
PRINT @r2;
```

ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
8134	16	1	usp_divide	8	Divide by zero error encountered.

RAISEERROR

```
RAISERROR ( { message_id | message_text |
@local_variable }
   { ,severity ,state }
   [ ,argument [ ,...n ] ] )
   [ WITH option [ ,...n ] ];
EXEC sp_addmessage
   @msgnum = 50005,
   @severity = 1,
   @msgtext = 'A custom error message';
```

RAISERROR

```
SELECT    * FROM    sys.messages
WHERE message_id = 50005;
RAISERROR ( 50005,1,1)
```

A custom error message

Msg 50005, Level 1, State 1

```
EXEC sp_dropmessage
   @msgnum = 50005;
```

message_text:

```
        RAISERROR ( 'Whoops, an error occurred.',1,1)
```

Whoops, an error occurred.

Msg 50000, Level 1, State 1

RAISERROR

Severity:

0–10 Informational messages

11–18 Errors

19–25 Fatal errors

State:

liczba o 0 do 255

WITH option:

- WITH LOG rejestruje błąd w dzienniku błędów i dzienniku aplikacji dla instancji serwera bazodanowego
- WITH NOWAIT natychmiast wysyła komunikat o błędzie do klienta
- WITH SETERROR ustawia wartości ERROR_NUMBER i @@ERROR na message_id lub 50000, niezależnie od poziomu ważności

RAISERROR z TRY CATCH

```
DECLARE
   @ErrorMessage  NVARCHAR(4000),
   @ErrorSeverity INT,
   @ErrorState    INT;

BEGIN TRY
   RAISERROR('Error occurred in the TRY block.', 17, 1);
END TRY
BEGIN CATCH
   SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
      @ErrorState = ERROR_STATE();

   -- return the error inside the CATCH block
   RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
END CATCH;
```

```
Msg 50000, Level 17, State 1, Line 16
Error occurred in the TRY block.
```

RAISERROR z dynamicznym tekstem wiadomości

RAISERROR statement:

```
DECLARE DECLARE @MessageText NVARCHAR(100);
SET @MessageText = N'Cannot delete the sales order %s';

RAISERROR(
   @MessageText, -- Message text
   16, -- severity
   1, -- state
   N'2001' -- first argument to the message text
);
MessageText NVARCHAR(100);
SET @MessageText = N'Cannot delete the sales order %s';

RAISERROR(
   @MessageText, -- Message text
   16, -- severity
   1, -- state
   N'2001' -- first argument to the message text
);
```

```
Msg 50000, Level 16, State 1, Line 5
Cannot delete the sales order 2001
```

RAISERROR – kiedy używać

- Rozwiązywanie problemów z kodem Transact-SQL
- Zwracanie wiadomości zawierającej tekst zmiennej
- Sprawdzanie wartości danych
- Powoduje, że wykonanie przeskakuje z bloku TRY do skojarzonego bloku CATCH
- Zwracanie informacji o błędach z bloku CATCH do wywołujących lub aplikacji