

Procedury tworzenie procedur przez użytkownika



Procedury

Procedura - jest to zestaw sekwencji poleceń do wykonania. Procedura może zwracać wartości wyprowadzone poprzez zmienne. W procedurze mogą zostać zawarte polecenia SQL DQL wyświetlające wyniki w ramach wykonania procedury (w przeciwieństwie do funkcji).

Procedura może zawierać w sobie również zestaw poleceń SQL DML, SQL TCL, jak również SQL DDL.

Wyświetlanie listy procedur: **show procedure status [like '%nazwa%']**

Wyświetlanie ciała procedury: **show create procedure [nazwa]**

Tabela opisująca procedury i funkcje: **information_schema.routines**

Procedury

podgląd charakterystyki procedury na podstawie tabeli
information_schema.routines

```
mariadb> desc information_schema.routines;
```

Field	Type	Null	Key	Default	Extra
SPECIFIC_NAME	varchar(64)	NO			
ROUTINE_CATALOG	varchar(512)	NO			
ROUTINE_SCHEMA	varchar(64)	NO			
ROUTINE_NAME	varchar(64)	NO			
ROUTINE_TYPE	varchar(13)	NO			
DATA_TYPE	varchar(64)	NO			
CHARACTER_MAXIMUM_LENGTH	int(21)	YES		NULL	
CHARACTER_OCTET_LENGTH	int(21)	YES		NULL	
NUMERIC_PRECISION	int(21)	YES		NULL	
NUMERIC_SCALE	int(21)	YES		NULL	
DATETIME_PRECISION	bigint(21) unsigned	YES		NULL	
CHARACTER_SET_NAME	varchar(64)	YES		NULL	
COLLATION_NAME	varchar(64)	YES		NULL	
DTD_IDENTIFIER	longtext	YES		NULL	
ROUTINE_BODY	varchar(8)	NO			
ROUTINE_DEFINITION	longtext	YES		NULL	
EXTERNAL_NAME	varchar(64)	YES		NULL	
EXTERNAL_LANGUAGE	varchar(64)	YES		NULL	
PARAMETER_STYLE	varchar(8)	NO			
IS_DETERMINISTIC	varchar(3)	NO			
SQL_DATA_ACCESS	varchar(64)	NO			
SQL_PATH	varchar(64)	YES		NULL	
SECURITY_TYPE	varchar(7)	NO			
CREATED	datetime	NO		0000-00-00 00:00:00	
LAST_ALTERED	datetime	NO		0000-00-00 00:00:00	
SQL_MODE	varchar(8192)	NO			
ROUTINE_COMMENT	longtext	NO			
DEFINER	varchar(189)	NO			
CHARACTER_SET_CLIENT	varchar(32)	NO			
COLLATION_CONNECTION	varchar(32)	NO			
DATABASE_COLLATION	varchar(32)	NO			

31 rows in set (0.13 sec)

Procedury

Uproszczony schemat tworzenia procedury:

```
CREATE  
[OR REPLACE]  
PROCEDURE sp_name ([proc_parameter[,...]])  
routine_body
```

proc_parameter:
[IN | OUT | INOUT] param_name type

type:
Any valid MariaDB data type
Any valid MySQL data type

routine_body:
Valid SQL procedure statement

<https://mariadb.com/kb/en/create-procedure/>
<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

Procedury

Rodzaje parametrów.

```
proc_parameter:  
[ IN | OUT | INOUT ] param_name type
```

W większości SZBD, procedury przyjmują następujące rodzaje parametrów:

IN - wyłącznie wprowadzamy wartości w ramach parametru, mogą to być liczby oraz zmienne wcześniej zadeklarowane z przypisanymi wartościami.

OUT - wyłącznie wyprowadzamy wartości z procedury poprzez parametr. W wywołaniu procedury parametr OUT nie może przyjmować liczb (czyli const), wyłącznie zmienne.

INOUT - wprowadzenie wartości zmiennej i wyprowadzanie zmodyfikowanej wartości przez procedurę. Nie może przyjmować liczb (czyli const), wyłącznie zmienne.

Procedury

Przykład - parametry typu IN:

```
delimiter ;;
```

```
CREATE PROCEDURE `dodaj_klienta`(IN vimie varchar(50),IN  
`vnazwisko` varchar(80))  
BEGIN  
if (char_length(vimie) > 0 and char_length(vnazwisko) > 0) then  
insert ignore into klient (imie, nazwisko) values  
(vimie,vnazwisko);  
end if;  
END
```

```
;;
```

```
delimiter ;
```

Procedurey

Przykład wywołania procedury w parametrach typu IN:

```
mysql> call dodaj_klienta('Michał', 'Nowak');  
Query OK, 1 row affected
```

Procedury

Przykład - parametry typu IN oraz OUT:

```
delimiter ;;
```

```
CREATE PROCEDURE wyswietl_klientow (IN `vnazwisko` varchar(80),  
OUT `vile` int)  
BEGIN
```

```
select imie, nazwisko from klient where nazwisko = vnazwisko;
```

```
set vile = FOUND_ROWS();
```

```
END
```

```
;;
```

```
delimiter ;
```

Funkcja FOUND_ROWS() zlicza
liczbę wyświetlonych poleceniem
select wierszy

Procedury

Przykład wywołania procedury z parametrami typu IN oraz OUT :

```
mysql> call wyswietl_klientow('Kowalski', @ile);
```

imie	nazwisko
Jan	Kowalski
D	Kowalski
M	Kowalski
F	Kowalski

4 rows in set

Query OK, 0 rows affected

```
mysql> select @ile;
```

@ile
4

Podstawy baz danych / Artur Niewiarowski

1 row in set

Procedury

Przykład - parametr typu INOUT:

```
delimiter ;;
```

```
CREATE PROCEDURE `duze_litery` (INOUT `wyrasz` varchar(100))  
BEGIN
```

```
set wyrasz = upper(wyrasz);
```

```
END
```

```
;;
```

```
delimiter ;
```

Procedury

Przykład wywołania procedury z parametrem typu INOUT:

```
mysql> call duze_litery(123);
```

1414 - OUT or INOUT argument 1 for routine artur.duze_litery is not a variable or NEW pseudo-variable in BEFORE trigger

```
mysql> set @info='politechnika krakowska';
```

Query OK, 0 rows affected

```
mysql> call duze_litery(@info);
```

Query OK, 0 rows affected

```
mysql> select @info;
```

@info

POLITECHNIKA KRAKOWSKA

1 row in set

Procedury

Przykład procedury z wywołaniem rekurencyjnym:

```
delimiter |

create procedure liczby (in liczba int)
begin

set liczba = liczba -1;

select liczba;

if liczba > 0 then
call liczby(liczba);
end if;

end |

delimiter ;
```

Efekt:

```
mysql> call liczby(4);
```

```
+-----+
| liczba |
+-----+
|      3 |
+-----+
1 row in set
```

```
+-----+
| liczba |
+-----+
|      2 |
+-----+
1 row in set
```

```
+-----+
| liczba |
+-----+
|      1 |
+-----+
1 row in set
```

```
+-----+
| liczba |
+-----+
|      0 |
+-----+
1 row in set
```

Query OK, 0 rows affected

Procedury

Wywołanie rekurencyjne ma miejsce po zmianie domyślnej wartości zmiennej globalnej *max_sp_recursion_depth* z 0 na 1 do 255.

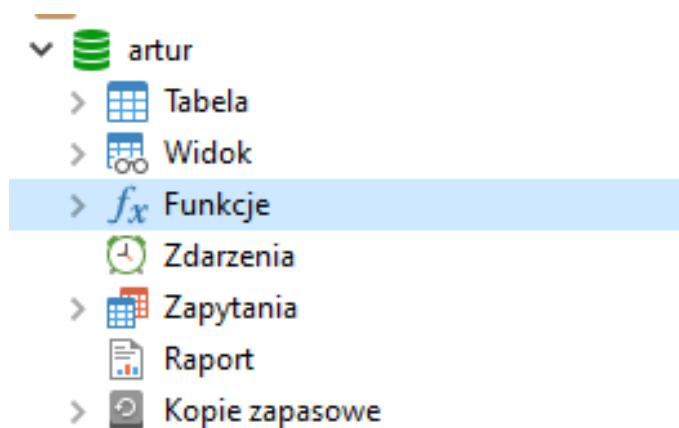
Np. na wartość 10

```
mysql> set @@max_sp_recursion_depth=10;  
Query OK, 0 rows affected
```

Więcej o zmiennych systemowych:
<https://mariadb.com/kb/en/server-system-variables/>

Procedury

program Navicat - kreator funkcji i procedur




Procedury

program Navicat - kreator funkcji i procedur

Procedury

program Navicat - kreator funkcji i procedur

 Kreator funkcji

Należy podać parametry procedury

Tryb	Nazwa	Typ
IN	vnazwisko	varchar
* OUT	vile	int

+

-

↑

↓

Opis

Tryb: określa, czy do parametru procedury zostanie przypisana wartość określona w procedurze.

Nazwa: określa nazwę parametru. To pole jest wymagane.

Typ: określa typ akceptowanych wartości. To pole jest wymagane.

☒ Używaj kreatora

< Wstecz

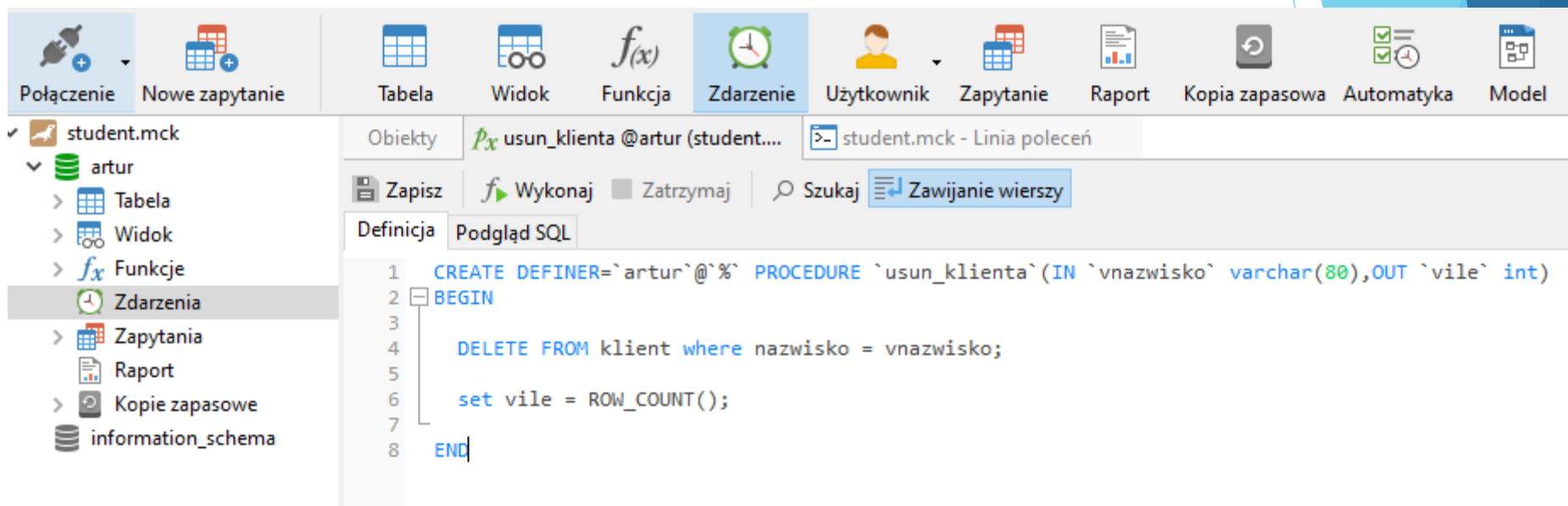
Dalej >

Zakończ

Anuluj

Procedury

program Navicat - kreator funkcji i procedur

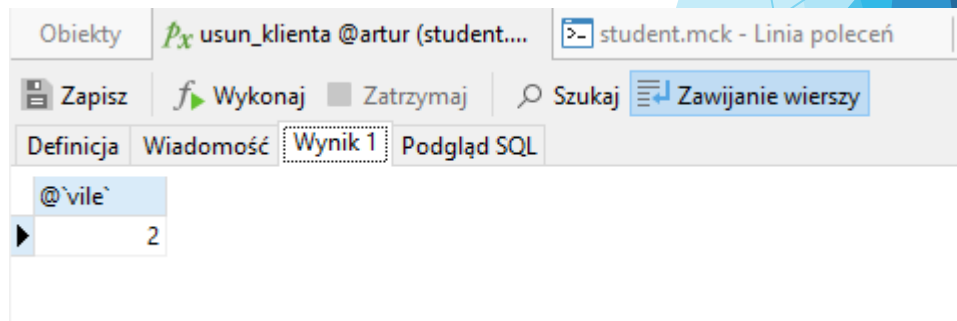
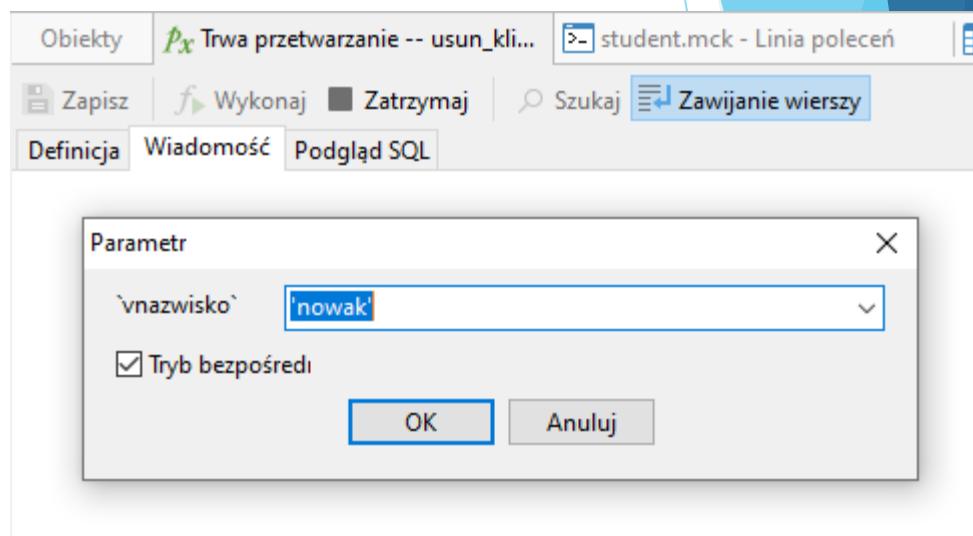


Procedury

program Navicat - kreator funkcji i procedur

```
mariadb> call usun_klienta('nowak', @ile);  
Query OK, 2 rows affected (0.03 sec)
```

```
mariadb> select @ile;  
+-----+  
| @ile |  
+-----+  
| 2    |  
+-----+  
1 row in set (0.03 sec)
```



Procedury - nadawanie uprawnień

```
GRANT CREATE ROUTINE ON baza_danych.* TO 'user'@'host';
```

```
--np.
```

```
mysql> GRANT CREATE ROUTINE ON artur.* TO 'artur'@'%';
```

```
Query OK, 0 rows affected
```

```
GRANT EXECUTE ON PROCEDURE baza_danych.procedura TO 'user'@'host';
```

```
--np.
```

```
mysql> GRANT EXECUTE ON PROCEDURE artur.usun_klienta TO 'artur'@'%';
```

```
Query OK, 0 rows affected
```

```
mysql> GRANT EXECUTE ON PROCEDURE artur.usun_klienta TO 'artur'@'localhost';
```

```
1133 - Can't find any matching row in the user table
```

Uwaga: aby nadać sobie dostęp do wykonania tylko konkretnej procedury w bazie danych (do której ma się pełne uprawnienia), należy najpierw poleceniem *revoke* odebrać uprawnienia do wszystkich procedur i następnie nadać zezwolenie do wybranej.

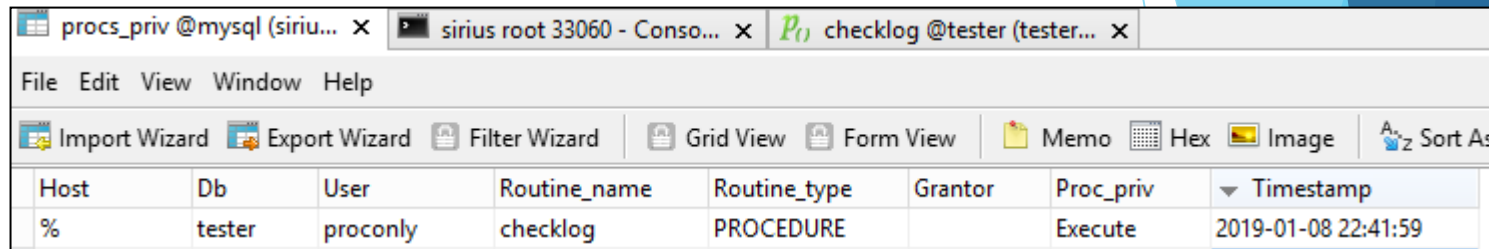
<https://mariadb.com/kb/en/grant/#procedure-privileges>

Procedury - nadawanie uprawnień

```
General DDL
CREATE DEFINER=`tester`@`%` PROCEDURE `checklog`(IN `vlogin` varchar(20),IN `vhaslo` varchar(40))
BEGIN

select count(*) cnt from users where login = vlogin and pass = vhaslo;

END
```



The screenshot shows the MySQL Workbench interface with the 'procs_priv' table selected. The table contains one row of data for the 'checklog' procedure.

Host	Db	User	Routine_name	Routine_type	Grantor	Proc_priv	Timestamp
%	tester	proconly	checklog	PROCEDURE		Execute	2019-01-08 22:41:59

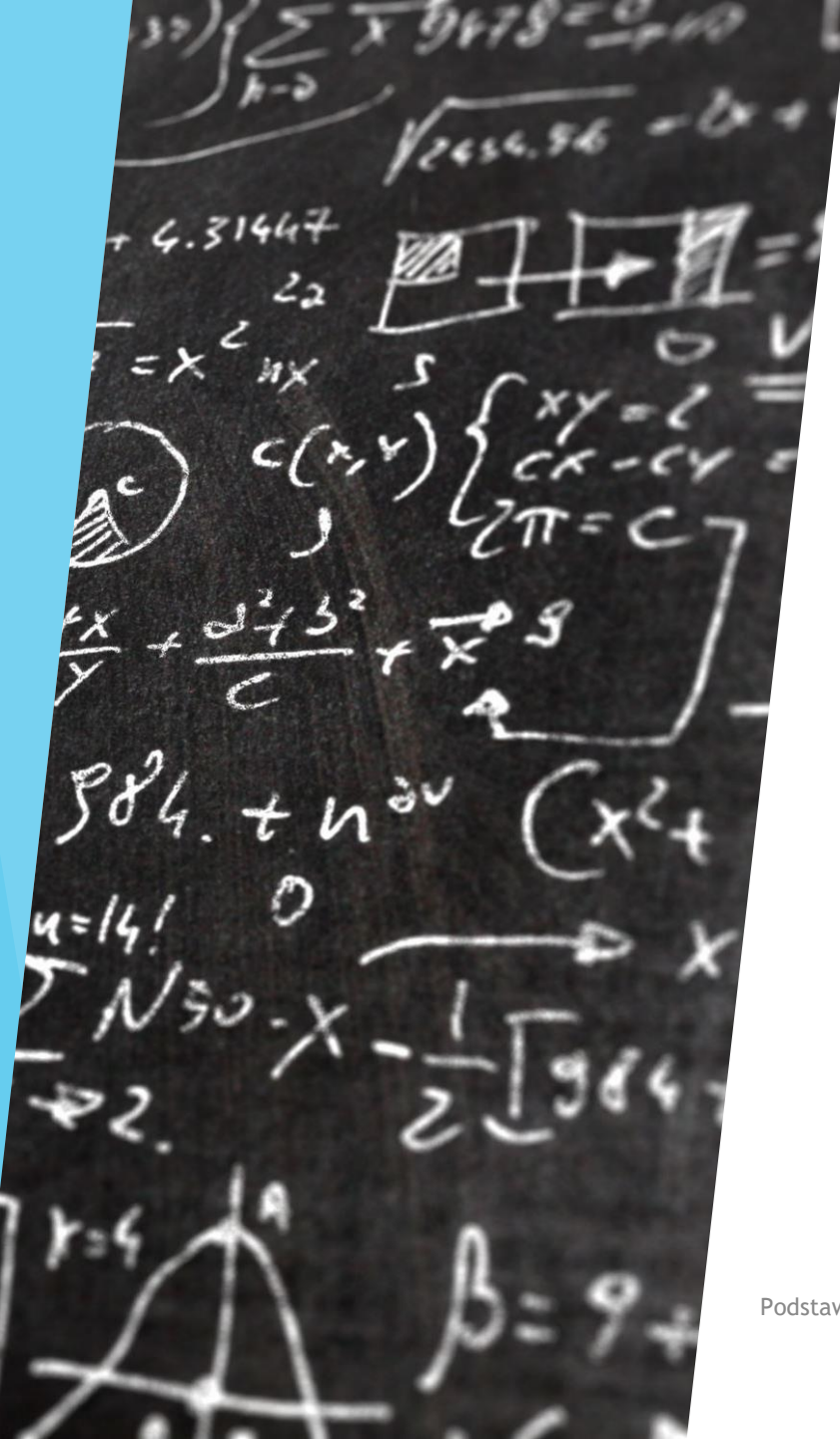
```
mysql> call checklog('jan', '123');
+-----+
| cnt |
+-----+
| 0 |
+-----+
1 row in set

Query OK, 0 rows affected

mysql> show tables;
Empty set

mysql> select * from tester.users;
1142 - SELECT command denied to user 'proconly'@'localhost' for table 'users'
mysql> |
```

Można stworzyć taką politykę bezpieczeństwa dostępu do danych w bazie danych, że dany użytkownik łączący się z poziomu środowiska programistycznego ma uprawnienia do wywoływania konkretnych procedur, ale nie ma dostępu do wykonania pozostałych poleceń SQL (*tj. select, update, itd.*), czyli również tych, które bezpośrednio w sobie zawiera procedura.



Funkcje tworzenie funkcji przez użytkownika

Funkcje

Funkcja - podobnie jak procedura, jest to zestaw sekwencji poleceń do wykonania. Funkcja zwraca wartość danego typu. Funkcja nie umożliwia wyświetlania wyników zapytań SQL umieszczonych w jej ciele- w przeciwieństwie do procedury.

W MariaDB od wersji 10.3.3 można tworzyć również funkcje agregujące. Tworzenie kodu funkcji można podzielić na dwa typy: standardowy domyślny MySQL/MariaDB oraz Oracle - PL/SQL - po ustawieniu zmiennej `sql_mode=Oracle`.

Wyświetlenie listy funkcji: **show function status [like '%nazwa%']**

Wyświetlenie ciała funkcji: **show create function [nazwa]**

Tabela opisująca procedury i funkcje: **information_schema.routines**

Funkcje

Uproszczony schemat tworzenia funkcji (w wersjach do 10.11):

```
CREATE [OR REPLACE]
[AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
RETURNS type
func_body
```

func_parameter:
param_name type

type:
Any valid MariaDB data type

func_body:
Valid SQL procedure statement

<https://mariadb.com/kb/en/create-function/>

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

Funkcje

Uproszczony schemat tworzenia funkcji (od wersji 11):

```
CREATE [OR REPLACE]
  [AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
  RETURNS type
  [characteristic ...]
  RETURN func_body
```

func_parameter:

```
[ IN | OUT | INOUT | IN OUT ] param_name type
```

type:

Any valid MariaDB data type

characteristic:

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| COMMENT 'string'
```

func_body:

Valid SQL procedure statement

Funkcje

Przykład tworzenia funkcji

```
CREATE FUNCTION `pole_prostokata`(`a` double, `b` double) RETURNS double
BEGIN
    #Routine body goes here...

    RETURN a*b;
END
```

- komentarz

` - ten znak nie jest konieczny, jeżeli nazwa nie zawiera spacji lub słów kluczowych dla polecenia

Funkcje

Kolejny przykład tworzenia funkcji

```
1 CREATE FUNCTION create (`a` double,`b` double) RETURNS double
2 BEGIN
3 RETURN a*b;
4 END
5
```

` - ten znak nie jest konieczny, jeżeli nazwa nie zawiera spacji lub słów kluczowych dla polecenia
Dotyczy to wszystkich obiektów tworzonych w BD

Message Summary Status

Query

```
CREATE FUNCTION create (`a` double,`b` double) RETURNS double
BEGIN
RETURN a*b;
END
```

Message

1064 - You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'create (`a` double,`b` double) RETURNS double BEGIN RETURN a*b; END' at line 1

arturn db.it

aniewiarowski123123

Run Stop Explain

```
1 CREATE FUNCTION `create` (`a` double,`b` double) RETURNS double
2 BEGIN
3 RETURN a*b;
4 END
5
```

Message Summary Profile Status

Query

```
CREATE FUNCTION `create` (`a` double,`b` double) RETURNS double
BEGIN
RETURN a*b;
END
```

Message

OK

Funkcje

```
delimiter ;;
```

```
CREATE FUNCTION `przelicz_klientow` (`vnazwisko` varchar(80))  
RETURNS int(11)  
BEGIN
```

```
declare ile int;  
select count(*) into ile from klient where nazwisko like  
concat('%',vnazwisko,'%');
```

```
RETURN ile;  
END;
```

```
;;
```

```
delimiter ;
```

Przykład utworzenia funkcji przez użytkownika za pomocą kodu SQL DDL

Delimiter - oznacza chwilową zmianę znaku kończącego polecenie SQL na inny (w tym przypadku jest to: ;;)

Związane jest to z tym, że w ciele funkcji występują znaki ; - bez zmiany delimitera pierwszy średnik zakończyłby tworzenie funkcji i wygenerowany zostałby błąd

Funkcje

```
mysql> select przelicz_klientow('kowalski');
```

```
+-----+  
| przelicz_klientow('kowalski') |  
+-----+  
| 1                               |  
+-----+
```

```
1 row in set
```

```
mysql> select przelicz_klientow('xkowalski');
```

```
+-----+  
| przelicz_klientow('xkowalski') |  
+-----+  
| 0                               |  
+-----+
```

```
1 row in set
```

```
mysql> select przelicz_klientow('kowals');
```

```
+-----+  
| przelicz_klientow('kowals') |  
+-----+  
| 2                             |  
+-----+
```

```
1 row in set
```

Przykład wywołania funkcji

Funkcje

Zadania:

1. Funkcja zliczająca liczbę użytkowników oprogramowania
2. Funkcje obliczające pole i obwód prostokąta
3. Funkcje obliczające pole i obwód trójkąta (wzór Herona)
4. Funkcje obliczające wartość minimalną i maksymalną z podanych
5. Funkcje obliczające pole i obwód trapezu
6. Funkcja obliczająca objętość walca
7. Funkcja obliczająca silnię
8. Funkcja sinus na bazie szeregu Taylora
9. Funkcja obliczająca n-ty wyraz ciągu liczb Catalana

Funkcje agregujące użytkownika

Schemat

```
CREATE AGGREGATE FUNCTION function_name (parameters) RETURNS return_type  
BEGIN
```

All types of declarations

```
DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN return_val;
```

```
LOOP
```

```
FETCH GROUP NEXT ROW; // fetches next row from table
```

other instructions

```
END LOOP;
```

```
END
```

<https://mariadb.com/kb/en/stored-aggregate-functions/>

Popularne funkcje agregujące zaimplementowane w systemie to:
min(), max(), avg(), sum() - omówione na poprzednich slajdach.

Więcej funkcji wbudowanych:

<https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html>

<https://mariadb.com/kb/en/aggregate-functions/>

Funkcje agregujące użytkownika

Przykład funkcji sumującej długości znaków w ciągach danej kolumny

```
DELIMITER //  
CREATE AGGREGATE FUNCTION IF NOT EXISTS suma_znakow(znaki varchar(100))  
RETURNS INT  
BEGIN  
    DECLARE suma INT DEFAULT 0;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN suma;  
    LOOP  
        FETCH GROUP NEXT ROW;  
        IF length(znaki) THEN  
            SET suma = suma+char_length(znaki);  
        END IF;  
    END LOOP;  
END //  
DELIMITER ;
```

Funkcje agregujące użytkownika

Przykład wywołania funkcji

```
mysql> select imie, nazwisko from klient where ID_klient <=3;
```

imie	nazwisko
Jan	Kowalski
Michał	Nowak
Anna	Kowalska

3 rows in set

```
mysql> select suma_znakow(imie) from klient where ID_klient <=3;
```

suma_znakow(imie)
13

1 row in set

Obsługa wyjątków

W MariaDB obsługa błędów jest realizowana za pomocą konstrukcji DECLARE HANDLER. Pozwala to na zdefiniowanie, jak ma zostać obsłużony określony rodzaj błędu lub wyjątku wewnątrz procedury składowanej lub bloku kodu.

Obsługa wyjątków

Struktura polecenia

```
DECLARE handler_type HANDLER
  FOR condition_value [, condition_value] ...
  statement
```

handler_type:

- CONTINUE
- | EXIT
- | UNDO

condition_value:

- SQLSTATE [VALUE] sqlstate_value
- | condition_name
- | SQLWARNING
- | NOT FOUND
- | SQLEXCEPTION
- | mariadb_error_code

```
DECLARE condition_name CONDITION FOR
condition_value
```

condition_value:

- SQLSTATE [VALUE] sqlstate_value
- | mysql_error_code

Obsługa wyjątków


1. `DECLARE HANDLER`: Rozpoczyna deklarację obsługi błędu.
2. `handler_type`: Określa rodzaj obsługi błędu. Może przyjmować jedną z trzech wartości:
 - `CONTINUE`: Kontynuuje wykonywanie programu po napotkaniu błędu.
 - `EXIT`: Wyjście z bloku kodu lub procedury po napotkaniu błędu.
 - `UNDO`: Cofnięcie transakcji i wyjście z bloku kodu lub procedury po napotkaniu błędu.
3. `condition_value`: Określa warunek, który musi być spełniony, aby obsłużyć błąd. Może to być jeden z następujących:
 - `SQLSTATE [VALUE] sqlstate_value`: Określa kod stanu SQL, który identyfikuje konkretny rodzaj błędu. Jest to pięciodziesiętny kod alfanumeryczny, gdzie pierwsze dwa znaki wskazują klasę błędu, a pozostałe trzy określają podklasę. Standard `sqlstate` opisany jest pod tym linkiem: <https://en.wikipedia.org/wiki/SQLSTATE>
 - `condition_name`: Nazwa zdefiniowanego przez użytkownika warunku błędu.
 - `SQLWARNING`, `NOT FOUND`, `SQLException`: Wbudowane warunki błędów oznaczające odpowiednio ostrzeżenie SQL, brak wyników i ogólny wyjątek SQL.
 - `mariadb_error_code`: Konkretny kod błędu MariaDB.
4. `statement`: Instrukcje, które mają zostać wykonane w przypadku wystąpienia błędu spełniającego warunek określony w `condition_value`.

Obsługa wyjątków

<https://en.wikipedia.org/wiki/SQLSTATE>

wikipedia.org/wiki/SQLSTATE

≡

WIKIPEDIA
The Free Encyclopedia

Search

SQLSTATE

[Add languages](#)

Contents hide

[\(Top\)](#)
[References](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#)

From Wikipedia, the free encyclopedia

Programs calling a database which accords to the [SQL](#) standard receive an indication about the success or failure of the call. This return code - which is called SQLSTATE - consists of 5 bytes. They are divided into two parts: the first and second bytes contain a **class** and the following three a **subclass**. Each class belongs to one of four **categories**: "S" denotes "Success" (class 00), "W" denotes "Warning" (class 01), "N" denotes "No data" (class 02) and "X" denotes "Exception" (all other classes).

- Real DBMSs are free to define additional values for SQLSTATE to handle those features which are beyond the standard. Such values must use one of the characters [I-Z] or [5-9] as the first byte of class (first byte of SQLSTATE) or subclass (third byte of SQLSTATE).
- In addition to SQLSTATE the SQL command `GET DIAGNOSTICS` offers more details about the last executed SQL command.
- In very early versions of the SQL standard the return code was called SQLCODE and used a different coding schema.

The following table lists the standard-conforming values - based on [SQL:2011](#).^[1] The table's last column shows the part of the standard that defines the row. If it is empty, the definition originates from part 2 *Foundation*.

SQLSTATE	Cat.	Class	Class Text	Subclass	Subclass Text	SQL part
00000	S	00	successful completion	000	(no subclass)	
01000	W	01	warning	000	(no subclass)	
01001	W	01	warning	001	cursor operation conflict	
01002	W	01	warning	002	disconnect error	
01003	W	01	warning	003	null value eliminated in set function	
01004	W	01	warning	004	string data, right truncation	

Obsługa wyjątków

<https://mariadb.com/kb/en/mariadb-error-codes/>

Shared MariaDB/MySQL error codes

Error Code	SQLSTATE	Error	Description
1018	HY000	ER_CANT_READ_DIR	Can't read dir of '%s' (errno: %d)
1019	HY000	ER_CANT_SET_WD	Can't change dir to '%s' (errno: %d)
1020	HY000	ER_CHECKREAD	Record has changed since last read in table '%s'
1021	HY000	ER_DISK_FULL	Disk full (%s); waiting for someone to free some space...
1022	23000	ER_DUP_KEY	Can't write; duplicate key in table '%s'
1023	HY000	ER_ERROR_ON_CLOSE	Error on close of '%s' (errno: %d)
1024	HY000	ER_ERROR_ON_READ	Error reading file '%s' (errno: %d)
1025	HY000	ER_ERROR_ON_RENAME	Error on rename of '%s' to '%s' (errno: %d)
1026	HY000	ER_ERROR_ON_WRITE	Error writing file '%s' (errno: %d)
1027	HY000	ER_FILE_USED	'%s' is locked against change
1028	HY000	ER_FILSORT_ABORT	Sort aborted

Obsługa wyjątków

przykład związany z transakcją bazodanową

```
CREATE PROCEDURE ExampleProcedure()  
BEGIN  
    DECLARE EXIT HANDLER FOR SQLEXCEPTION  
    BEGIN  
        -- Obsługa wyjątków SQL  
        ROLLBACK;  
    END;  
    START TRANSACTION;  
    -- Kod, który może generować wyjątki  
    COMMIT;  
END;
```

Obsługa wyjątków

przykład użycia SQLSTATE

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `dodaj_uzytkownika`(vlogin
VARCHAR(100), vhaslo VARCHAR(200), vimie varchar(100), vnazwisko varchar(100))
BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE '23000'
    BEGIN
        -- Zapisanie błędu i poinformowanie użytkownika o problemie
        INSERT INTO log_errors (message) VALUES (concat('Próba dodania
użytkownika \'', vlogin, '\', z naruszeniem indeksu unique.'));
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Nie można dodać
użytkownika - podany login już istnieje.';
    END;

    -- Próba dodania użytkownika, która może spowodować błąd 23000
    INSERT INTO uzytkownicy (login, haslo, imie, nazwisko) VALUES (vlogin,
vhaslo, vimie, vnazwisko);
END
```

Jeżeli operacja nie powiedzie się z powodu naruszenia ograniczenia klucza obcego - SQLSTATE dla tego rodzaju błędu to 23000, co wskazuje na błąd związany z naruszeniem integralności.

Obsługa wyjątków

przykład użycia SQLSTATE

Wysyłany jest dalszy sygnał z SQLSTATE '45000', który jest ogólnym kodem błędu dla błędów użytkownika, z odpowiednią wiadomością

44000	X	44	with check option violation	000	(no subclass)	
45000	X	45	unhandled user-defined exception	000	(no subclass)	SQL/PSM

Objects | uzytkownicy @artur (root local) - Table | P_x dodaj_uzytkownika @artur (root local) - ... | root local - Console

```
mariadb> call dodaj_uzytkownika('anowak', '123', 'Anna', 'Nowak');
Query OK, 1 row affected (0.00 sec)

mariadb> call dodaj_uzytkownika('anowak', '123', 'Anna', 'Nowak');
1644 - Nie można dodać użytkownika - podany login już istnieje.
mariadb> show errors;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Error | 1644 | Nie można dodać użytkownika - podany login już istnieje. |
+-----+-----+-----+
1 row in set (0.03 sec)

mariadb>
```

1644	HY000	ER_SIGNAL_EXCEPTION	Unhandled user-defined not found condition
1644	HY000	ER_SIGNAL_EXCEPTION	Unhandled user-defined exception condition

Obsługa wyjątków

SQLSTATE a ERROR CODE

SQLSTATE i Error Code to dwa różne sposoby reprezentacji błędów w systemach baz danych, takich jak MariaDB. Obie metody mają swoje zastosowania i różnią się pod kilkoma kluczowymi względami:

SQLSTATE

- Standaryzacja: SQLSTATE jest standardem ANSI i ISO, który jest wspólny dla wielu systemów zarządzania bazami danych. Dzięki temu kod SQLSTATE jest bardziej przenośny między różnymi systemami baz danych.
- Format: SQLSTATE to pięciodziesiętny kod alfanumeryczny, gdzie pierwsze dwa znaki reprezentują klasę błędu, a trzy pozostałe - podklasę. Ten format pozwala na szybkie zidentyfikowanie kategorii błędu.
- Abstrakcja: Kody SQLSTATE są bardziej ogólne, co oznacza, że te same kody mogą być używane w różnych kontekstach błędów, choć odnoszą się do podobnych kategorii problemów.

Obsługa wyjątków

SQLSTATE a ERROR CODE

SQLSTATE i Error Code to dwa różne sposoby reprezentacji błędów w systemach baz danych, takich jak MariaDB. Obie metody mają swoje zastosowania i różnią się pod kilkoma kluczowymi względami:

Error Code

- Specyfika systemu: Error Codes są specyficzne dla danego systemu baz danych i mogą dostarczać bardziej szczegółowych informacji o błędzie. Na przykład, MariaDB ma własny zestaw kodów błędów, które nie zawsze są zgodne z kodami w innych systemach, takich jak Oracle czy PostgreSQL.
- Format: Error Codes w MariaDB to zazwyczaj liczby całkowite, które są bezpośrednio powiązane z konkretnym typem błędu.
- Szczegółowość: Kody błędów są zwykle bardziej szczegółowe niż kody SQLSTATE i mogą dostarczać specyficzne informacje potrzebne do diagnozy i rozwiązania problemu.

Obsługa wyjątków

SQLSTATE a ERROR CODE

Przykład

Założmy, że operacja na bazie danych MariaDB generuje błąd z powodu duplikacji klucza. SQLSTATE dla tego błędu może być 23000, co wskazuje na błąd integralności. W tym samym czasie, MariaDB może przypisać konkretny Error Code, na przykład 1062, który dokładnie określa, że doszło do naruszenia ograniczenia unikalności klucza.

Podsumowując, SQLSTATE zapewnia bardziej ogólną klasyfikację błędów zgodnie z międzynarodowymi standardami, natomiast Error Code dostarcza bardziej szczegółowych informacji związanych bezpośrednio z implementacją konkretnej bazy danych. W praktyce, dobrze jest używać obu kodów do diagnozy problemów, gdzie SQLSTATE może służyć do obsługi błędów na wyższym poziomie abstrakcji, a Error Code do dokładnego zrozumienia i

1048	23000	ER_BAD_NULL_ERROR	Column '%s' cannot be null
1049	42000	ER_BAD_DB_ERROR	Unknown database '%s'
1050	42S01	ER_TABLE_EXISTS_ERROR	Table '%s' already exists
1051	42S02	ER_BAD_TABLE_ERROR	Unknown table '%s'
1052	23000	ER_NON_UNIQ_ERROR	Column '%s' in '%s' is ambiguous
1053	08S01	ER_SERVER_SHUTDOWN	Server shutdown in progress
1054	42S22	ER_BAD_FIELD_ERROR	Unknown column '%s' in '%s'
1055	42000	ER_WRONG_FIELD_WITH_GROUP	'%s' isn't in GROUP BY
1056	42000	ER_WRONG_GROUP_FIELD	Can't group on '%s'
1057	42000	ER_WRONG_SUM_SELECT	Statement has sum functions and columns in same statement
1058	21S01	ER_WRONG_VALUE_COUNT	Column count doesn't match value count
1059	42000	ER_TOO_LONG_IDENT	Identifier name '%s' is too long
1060	42S21	ER_DUP_FIELDNAME	Duplicate column name '%s'
1061	42000	ER_DUP_KEYNAME	Duplicate key name '%s'
1062	23000	ER_DUP_ENTRY	Duplicate entry '%s' for key '%d'

Obsługa wyjątków

przykład użycia SQLSTATE

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `dodaj_uzytkownika`(vlogin
VARCHAR(100), vhaslo VARCHAR(200), vimie varchar(100), vnazwisko varchar(100))
BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE '23000'
    BEGIN
        -- Zapisanie błędu i poinformowanie użytkownika o problemie
        INSERT INTO log_errors (message) VALUES (concat('Próba dodania
użytkownika \'', vlogin, '\', z naruszeniem indeksu unique.'));
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Nie można dodać
użytkownika - podany login już istnieje.', mysql_errno = 45123;
    END;

    -- Próba dodania użytkownika, która może spowodować błąd 23000
    INSERT INTO uzytkownicy (login, haslo, imie, nazwisko) VALUES (vlogin,
vhaslo, vimie, vnazwisko);
END
```

, mysql_errno = 45123; - definiujemy własny kod błędu „ERROR CODE”

Obsługa wyjątków

przykład użycia SQLSTATE

```
CREATE D  
VARCHAR(  
BEGIN  
DECL  
BEGI  
  
użytkown  
użytkown  
END;  
  
-- P  
INSE  
vhasło,  
END
```

```
mariadb> call dodaj_uzytkownika('anowak', '123', 'Anna', 'Nowak');  
Query OK, 1 row affected (0.00 sec)
```

```
mariadb> call dodaj_uzytkownika('anowak', '123', 'Anna', 'Nowak');  
1644 - Nie można dodać użytkownika - podany login już istnieje.  
mariadb> show errors;
```

Level	Code	Message
Error	1644	Nie można dodać użytkownika - podany login już istnieje.

```
1 row in set (0.03 sec)
```

```
mariadb> call dodaj_uzytkownika('anowak', '123', 'Anna', 'Nowak');  
45123 - Nie można dodać użytkownika - podany login już istnieje.
```

```
mariadb> show errors;
```

Level	Code	Message
Error	45123	Nie można dodać użytkownika - podany login już istnieje.

```
1 row in set (0.03 sec)
```

```
mariadb> |
```

login
varchar(100))

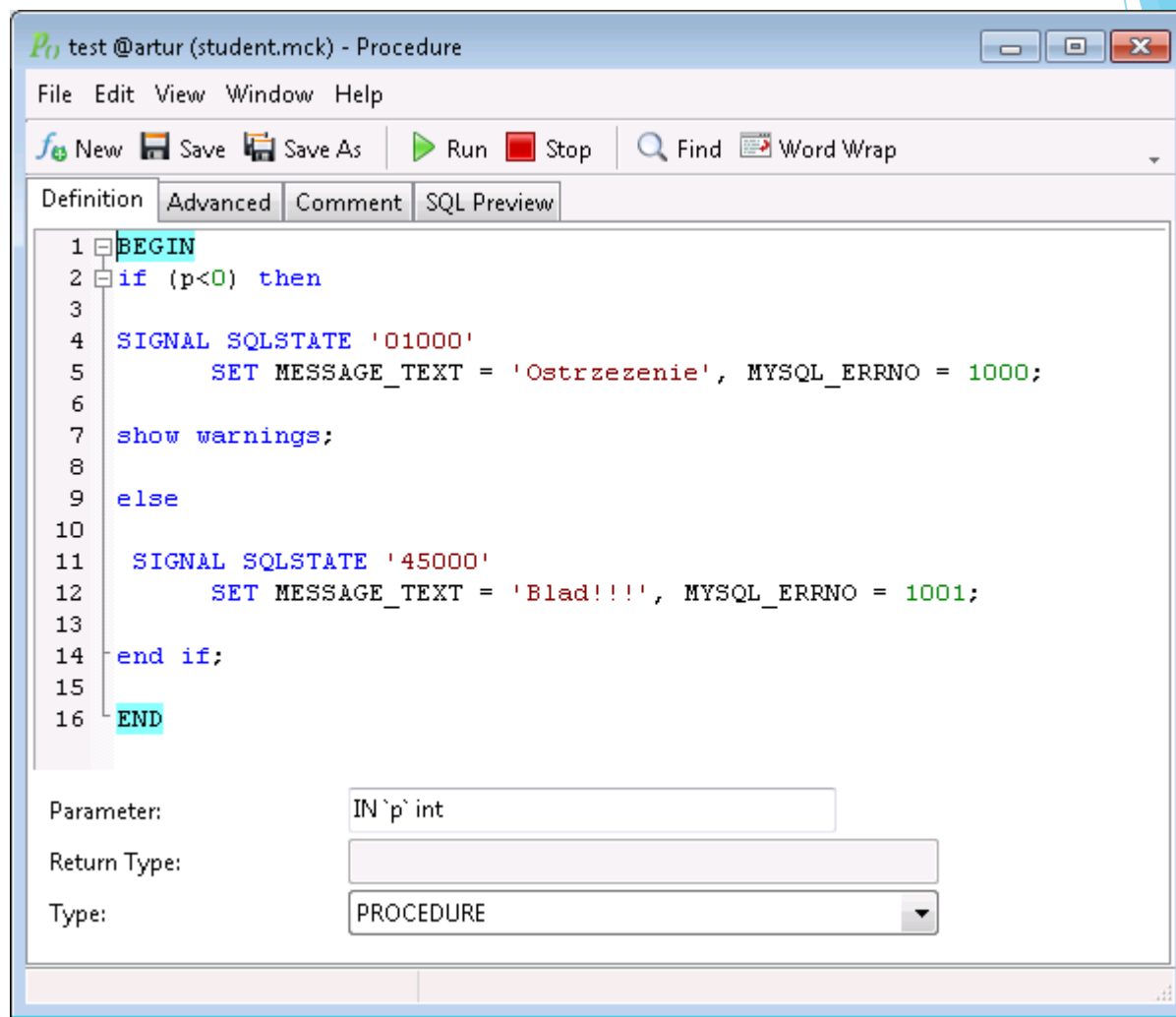
nie
podania

łać

00
(vlogin,

, mysql_errno = 45123; - definiujemy własny kod błędu „ERROR CODE”

Obsługa wyjątków

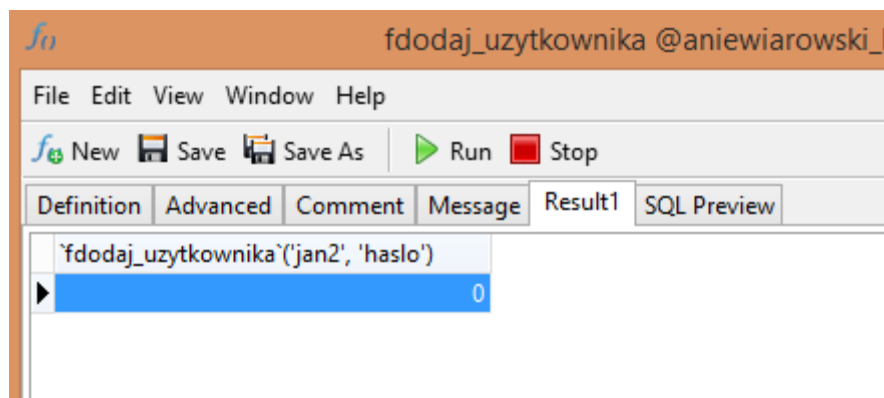
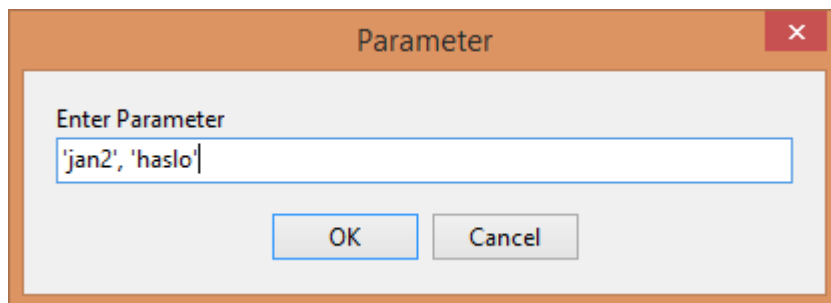


Funkcje-Procedury

Obsługa wyjątków

<https://mariadb.com/kb/en/declare-handler/>

Funkcje - program Navicat



Elementy języka programowania w Systemach Zarządzania Bazami Danych

w oparciu o
SQL/PSM
Oracle PL/SQL
w MySQL i MariaDB

Elementy języka programowania

- SQL/PSM - skrót od: SQL/Persistent Stored Modules. Jako standard ISO powstał ok. 1996 roku i oznacza rozszerzenie języka SQL o możliwość programowania proceduralnego.

Systemy, które go obsługują to m.in.: MySQL, MariaDB, PostgreSQL, Oracle.

- PL/SQL - skrót od: Procedural Language for SQL. Jest to rozszerzenie języka SQL przeznaczone dla SZBD Oracle. Również powstał w latach '90. Implementuje w pewnym stopniu standard SQL/PSM.

Systemy, które go obsługują to m.in. Oracle (jako system macierzysty), MariaDB (posiada kompatybilny parser), PostgreSQL (umożliwia poprzez emulator emulowanie w pewnym stopniu języka).

Elementy języka programowania

Wiele zaawansowanych klientów bazodanowych posiada funkcjonalności ułatwiające programowanie proceduralne w bazach danych, np. Navicat (poniżej).

The screenshot displays the Navicat SQL editor interface. The main window shows a SQL script for a procedure named 'p1'. The script includes a REPEAT loop, a WHILE loop, an IF-THEN-ELSE conditional, and a CASE statement. The sidebar on the right, titled 'Kontrola przepływu' (Flow Control), lists five constructs: CASE, IF...ELSE..., LOOP, REPEAT, and WHILE, each with a brief description of its function. The bottom status bar indicates the user is working on 'Podstawy baz danych' (Database Fundamentals) by 'Artur Niewiarowski'.

Obiekty *Px* * Bez nazwy @artur (student.m...)

Zapisz Wykonaj Zatrzymaj Szukaj Zawijanie wierszy

Definicja Podgląd SQL

```
1 CREATE DEFINER = CURRENT_USER PROCEDURE `p1`()
2 BEGIN
3     #Routine body goes here...
4 REPEAT
5     statement_list
6 UNTIL search_condition END REPEAT;
7
8 WHILE search_condition DO
9     statement_list
10 END WHILE;
11
12 IF search_condition THEN
13     statement_list
14 ELSE
15     statement_list
16 END IF;
17
18 CASE case_value
19     WHEN when_value THEN
20         statement_list
21     ELSE
22         statement_list
23 END CASE;
```

Podstawy baz danych / Artur Niewiarowski

Kontrola przepływu

- CASE** Kontrola przepływu
Create a conditional construct
- IF...ELSE...** Kontrola przepływu
Create a IF...ELSE... construct
- LOOP** Kontrola przepływu
Create a simple loop construct
- REPEAT** Kontrola przepływu
Create A REPEAT construct. The Statement list is repeated until the search_condition expression is true.
- WHILE** Kontrola przepływu
Create a WHILE construct. The statement list within a WHILE statement is repeated as long as the search_condition expression is true.

Szukaj

Elementy języka programowania

Warunek IF - składnia

```
CREATE DEFINER=CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
IF search_condition THEN  
    statement_list  
ELSE  
    statement_list  
END IF;
```

```
END;
```

Elementy języka programowania

Warunek IF - przykład

```
CREATE DEFINER=`artur`@`%` PROCEDURE `test`()  
BEGIN
```

```
DECLARE x int DEFAULT 5;  
DECLARE y int DEFAULT 6;  
DECLARE w int;
```

```
IF x>0 and y>0 or x<=y THEN  
set w=x+y;  
ELSEIF x=0 THEN  
set w:=x-y;  
ELSE  
set w:=x*y;  
END IF;
```

```
select w;  
END
```

Elementy języka programowania

Warunek IF - przykład

```
CREATE DEFINER=`artur`@`%` FUNCTION `test`() RETURNS int(11)  
BEGIN
```

```
DECLARE x int DEFAULT 5;  
DECLARE y int DEFAULT 6;  
DECLARE w int;
```

```
IF x>0 and y>0 or x<=y THEN  
set w:=x+y;  
ELSEIF x=0 THEN  
set w:=x-y;  
ELSE  
set w:=x*y;  
END IF;
```

```
return w;  
END
```

Elementy języka programowania

Wybór CASE - składnia

```
CREATE DEFINER=`artur`@`%` PROCEDURE `test`()  
BEGIN
```

```
    CASE case_value  
        WHEN when_value THEN  
            statement_list  
        ELSE  
            statement_list  
    END CASE;
```

```
END
```

Elementy języka programowania

Wybór CASE - przykład

```
CREATE DEFINER=`artur`@`%` PROCEDURE `test`()  
BEGIN
```

```
DECLARE x int DEFAULT 5;  
DECLARE y int DEFAULT 6;
```

```
CASE x+y  
  WHEN 1 THEN  
select CONCAT('Wybrano: ',x+y);  
  WHEN 2 THEN  
  select 'Wybrano: 2';  
  ELSE  
  select 'Wybrano: 11';  
END CASE;
```

```
END
```


Elementy języka programowania

Pętla LOOP - składnia

```
CREATE DEFINER=CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
moja_petla: LOOP  
    statement_list  
  
    IF exit_condition THEN  
        LEAVE moja_petla;  
    END IF;  
END LOOP moja_petla;
```

```
END;
```

Elementy języka programowania

Pętla LOOP - przykład

```
CREATE DEFINER=`artur`@`%` PROCEDURE `silnia`(IN x int)
BEGIN
    set @k=1;
    set @w=1;
    label: LOOP

    IF @k >= x THEN
        LEAVE label;
    ELSE
        set @k:=@k+1;
        set @w:=@w*@k;

    END IF;
END LOOP label;

select @w as wynik;

END
```

Elementy języka programowania

Deklaracja zmiennych

```
CREATE PROCEDURE prc_test ()  
BEGIN  
    DECLARE var2 INT DEFAULT 1;  
    SET var2 = var2 + 1;  
    SET @var2 = @var2 + 1;  
    SELECT var2, @var2;  
END;
```

@var2 - zmienna sesyjna, inicjalizowana raz, widoczna również poza funkcją lub procedurą

var - zmienna reinicjalizowana, za każdym razem, gdy wywoływana jest w ramach funkcji lub procedury

```
SET @var2 = 1;
```

```
CALL prc_test();
```

var2	@var2
---	---
2	2

```
CALL prc_test();
```

var2	@var2
---	---
2	3

```
CALL prc_test();
```

var2	@var2
---	---
2	4

Elementy języka programowania

Pętla LOOP - przykład

```
mariadb> call silnia(5);
```

```
+-----+
```

```
| wynik |
```

```
+-----+
```

```
| 120   |
```

```
+-----+
```

```
1 row in set (0.04 sec)
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mariadb>
```

Elementy języka programowania

Pętla WHILE - składnia

```
CREATE DEFINER = CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
    WHILE search_condition DO  
        statement_list  
    END WHILE;
```

```
END;
```

Elementy języka programowania

Pętla WHILE - przykład

```
CREATE DEFINER=`artur`@`%`—PROCEDURE `silnia2`(IN x int)
BEGIN

  set @w := 1;
  WHILE x>0 DO
    set @w:=@w * x;
  set x := x-1;
  END WHILE;

  select @w;

END
```

Elementy języka programowania

Pętla REPEAT - składnia

```
CREATE DEFINER = CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
    REPEAT  
        statement_list  
    UNTIL search_condition END REPEAT;
```

```
END;
```

Elementy języka programowania

Pętla REPEAT - przykład

```
CREATE DEFINER=`artur` `@`%` FUNCTION `silnia3` (`x` int) RETURNS  
int(11)  
BEGIN  
  
if x<1 then set x:=1; end if;  
  
set @w:=1;  
  REPEAT  
    set @w:=@w*x;  
    set x:=x-1;  
  UNTIL x<=1 END REPEAT;  
  
  RETURN @w;  
END
```


Elementy języka programowania

Pętla REPEAT - przykład

```
mariadb> select silnia3(5);
```

```
+-----+
```

```
| silnia3(5) |
```

```
+-----+
```

```
| 120 |
```

```
+-----+
```

```
1 row in set (0.06 sec)
```

Elementy języka programowania

Pętla FOR - składnia

```
CREATE DEFINER = CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
FOR  
    var_name IN [REVERSE] lower_bound .. upper_bound  
DO statement_list
```

```
END FOR;  
END;
```

Elementy języka programowania

Pętla FOR - składnia

```
CREATE OR REPLACE FUNCTION `silnia_for`(`x` bigint) RETURNS  
bigint(20)  
BEGIN
```

```
declare wynik bigint default 1;
```

```
for k in 1 .. x do  
set wynik = wynik * k;  
end for;
```

```
RETURN wynik;  
END
```

```
mariadb> select silnia_for(5);  
+-----+  
| silnia_for(5) |  
+-----+  
|              120 |  
+-----+  
1 row in set (0.02 sec)
```

Elementy języka programowania

Pętla FOR - składnia

```
CREATE DEFINER=CURRENT_USER PROCEDURE `test`()  
BEGIN
```

```
FOR  
    record_name IN (select_statement)  
DO statement_list
```

```
END FOR;  
END;
```

Elementy języka programowania

Pętla FOR - przykład

```
CREATE PROCEDURE `proc0`(in vlogin varchar(50))
BEGIN
  for rekord in (select * from uzytkownicy where lower(login) =
lower(vlogin))
  do
    insert into users (login, haslo, imie, nazwisko)
    values (rekord.login, rekord.haslo, 'nie podano', 'nie podano');
  end for;
END
```

Elementy języka programowania

Przykłady wywołania funkcji

```
select silnia(5);
```

```
select * from liczby where x > silnia(y);
```

```
insert into liczby (x,y) values (x, silnia(5));
```

```
delete liczby where y < silnia(x);
```

```
update liczby set y = silnia(x);
```

Elementy języka programowania

Przykłady wywołania funkcji

```
CREATE FUNCTION `tangens`(`x` double) RETURNS double  
BEGIN  
  
    RETURN sin(x)/cos(x);  
END
```

Funkcje możemy również wywoływać w innych funkcjach oraz procedurach