

Procedury i funkcje w MySQL.

Tworzenie kopii zapasowej w MySQL.

Transakcje- ćwiczenie

Oprócz tabel i widoków w bazach danych możemy tworzyć własne funkcje, procedury i specjalny typ procedur, które będą automatycznie wywoływane przez MySQL w momencie wstawiania, usuwania lub modyfikowania określonych danych, czyli wyzwalacze.

Funkcje użytkownika

Jedną z możliwości MySQL-a jest możliwość tworzenia przez użytkowników własnych funkcji. Funkcjom, tak jak procedurom, można przekazać pewną liczbę parametrów, ale funkcja nie tylko wykonuje pewne operacje, ale także zwraca obliczony na podstawie przekazanych parametrów wynik.

Funkcje różnią się od procedur tym, że nie muszą być wskazywane parametry wejścia oraz wyjścia (funkcje bezparametrowe). Ponadto funkcja zwraca jakiś typ danych z dostępnych w SQL.

Ogólna postać:

```
CREATE FUNCTION nazwa_f(parametry) RETURNS typ
BEGIN
    -- polecenia
    RETURN wynik
END
```

Tworzenie funkcji

Wykonanie instrukcji CREATE FUNCTION spowoduje utworzenie funkcji użytkownika. Funkcja z przykładu oczekuje na jeden parametr — datę — i zwraca odczytany z tej daty rok.

```
CREATE FUNCTION fn_data
(data DATE)
RETURNS CHAR(20)
RETURN YEAR(data);
```

Osoby znające inne proceduralne języki programowania prawdopodobnie rozpoznają pewne podobieństwa w tworzeniu funkcji — w każdym przypadku trzeba zdefiniować nagłówek (nazwę, listę argumentów i typ zwracanej wartości) i ciało funkcji. Ostatnią instrukcją ciała funkcji powinna być instrukcja RETURN. Funkcja zwróci wartości wymienione po prawej stronie tej instrukcji.

Również można wykonać funkcję w formie graficznej:

fn_data

Szczegóły

Nazwa procedury: fn_data

Typ: FUNCTION

Parametry	Nazwa	Typ	Długość/Wartości	Opcje
	data	DATE	---	---

Dodaj parametr

Zwracany typ: CHAR

Powrót długości/wartości: 20

Powrót opcji: Kodowanie znaków

Określenie

```
1 RETURN YEAR(data)
```

Jest deterministyczna: ☐

Dostosuj uprawnienia: ☒

Zastosowanie:

```
SELECT `data_start_rezerwacji`,fn_data(data_koniec_rezerwacji) FROM `rezerwacje`;
```

		data_start_rezerwacji	fn_data(data_koniec_rezerwacji)
☐	Edytuj	Kopiuje	Usuń
		2016-05-12	2016
☐	Edytuj	Kopiuje	Usuń
		2016-05-14	2016

Przykład 2:

```
CREATE FUNCTION fn_daty
(
  data date, delimiter char(1)
)
RETURNS char(20)
RETURN CONCAT(YEAR(data),delimiter, MONTHNAME(data));
```

Podczas korzystania z funkcji mogą wystąpić pewne problemy.

Jednym z ważniejszych jest znak średnika ';', który jest wpisywany podczas tworzenia funkcji/procedury. Jest prosty sposób na obejście tego. Wystarczy, że wpiszemy np. "DELIMITER //", co spokojnie pozwoli nam na zapisanie kodu. Natomiast znak średnika zastąpi '//'. Tak więc warto nie zapominać o tym, a na końcu warto wpisać "DELIMITER ;" tak aby średnik miał swoje dawne "znaczenie".

fn_daty

Szczegóły

Nazwa procedury: fn_daty

Typ: FUNCTION

Parametry	Nazwa	Typ	Długość/Wartości	Opcje
↑	data	DATE	---	---
↑	delimiter	CHAR	1	Kodowa

Dodaj parametr

Zwracany typ: CHAR

Powrót długości/wartości: 20

Powrót opcji: Kodowanie znaków

```
1 RETURN CONCAT(YEAR(data),delimiter, MONTHNAME(data))
```

Wywołanie na przykładzie bazy hotel:

```
SELECT data_start_rezerwacji,fn_daty(data_start_rezerwacji,':') FROM rezerwacje
```

			data_start_rezerwacji	fn_daty(data_start_rezerwacji,':')
☐		Edytuj		Kopiuj
☐		Usuń	2016-05-12	2016:May
☐		Edytuj		Kopiuj
☐		Usuń	2016-05-14	2016:May

Usuwanie funkcji

Wykonanie instrukcji DROP FUNCTION spowoduje usunięcie wybranych funkcji użytkownika

```
DROP FUNCTION fn_data;
```

Procedura

Procedura jest podprogramem utworzonym w SQL, która zawiera kod SQL oraz kod sterujący jej wykonaniem.

Procedury są przydatne wtedy, gdy chcemy wykonać jakieś często powtarzające się operacje.

Postać procedury:

```
CREATE PROCEDURE nazwa_procedury (IN zmienna typ, OUT zmienna typ)
BEGIN
    -- polecenia i kod SQL
END
```

Przykład użycia procedury, obniżenie ceny o 30%

```
CREATE PROCEDURE `obnizka30` (IN `nazwa` VARCHAR(50)) NOT DETERMINISTIC NO SQL SQL SECURITY DEFINER UPDATE produkt SET cena = cena * 0.7 WHERE nazwa = nazwa
```

Nazwa procedury	obnizka30			
Typ	PROCEDURE			
Parametry	Kierunek	Nazwa	Typ	Długość
	IN	nazwa	VARCH.	50
Dodaj parametr				
<pre>1 UPDATE produkt 2 SET cena = cena * 0.7 3 WHERE nazwa = nazwa</pre>				

Wywołanie procedury:

```
call obnizka30('mikser');
```

Procedura dodawania klienta: na bazie sklepik_abc:

dodaj_klienta

Szczegóły

Nazwa procedury

dodaj_klienta

Typ

PROCEDURE

Parametry

Kierunek	Nazwa	Typ	Długość/Wartości	Opcje
IN	klient_id	INT		Usun
IN	imie	VARC	50	Ko Usun
IN	nazwisko	VARC	50	Ko Usun
IN	miasto	VARC	50	Ko Usun
IN	kraj	VARC	50	Ko Usun
IN	telefon	CHAR	12	Ko Usun

Dodaj parametr

```
1 INSERT INTO klienci(klient_id, imie, nazwisko, miasto, kraj, telefon) VALUES (null,imie,nazwisko,miasto, kraj, telefon)
```

Wykonaj

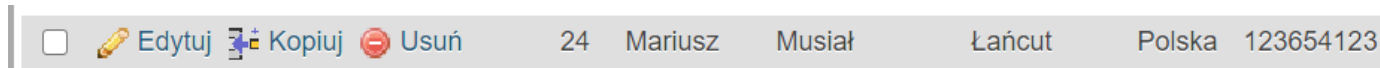
Zamknij

```
CREATE PROCEDURE `dodaj_klienta` (IN `klient_id` INT, IN `imie` VARCHAR(50), IN `nazwisko` VARCHAR(50), IN `miasto` VARCHAR(50), IN `kraj` VARCHAR(50), IN `telefon` CHAR(12)) NOT DETERMINISTIC NO SQL SQL SECURITY DEFINER INSERT INTO klienci(klient_id, imie, nazwisko, miasto, kraj, telefon) VALUES (null,imie,nazwisko,miasto, kraj, telefon)
```

Wywołanie procedury dodaj_klienta:

```
CALL dodaj_klienta(null, 'Mariusz', 'Musiał', 'Łańcut', 'Polska', 123654123);
```

Dodany klient, przez procedurę:



Usuwanie procedur

Nieużywane procedury możemy usunąć poprzez instrukcję **DROP PROCEDURE nazwa_procedury;**

Często podczas wykonywania pewnych operacji na tabelach w bazie danych, takich jak wstawianie, usuwanie czy aktualizowanie rekordów, trzeba zaktualizować inną tabelę i/lub wykonać pewne obliczenia za każdym razem, gdy modyfikujemy dane. Można wtedy używać triggerów, czyli wyzwalaczy.

Triggerem (wyzwalaczem) nazywamy procedurę, która jest połączona z konkretną tabelą w bazie danych i wywołuje się automatycznie jako reakcja na wcześniej określone zdarzenie występujące w tej tabeli.

Do zdarzeń, które powodują wywołanie triggera, zalicza się:

- INSERT - dodanie nowego rekordu,
- UPDATE - zmodyfikowanie istniejącego rekordu,
- DELETE - usunięcie rekordu z bazy danych.

Trigger może być ustawiony do wywołania przed (BEFORE) lub po (AFTER) zdarzeniu, które go wywołuje.

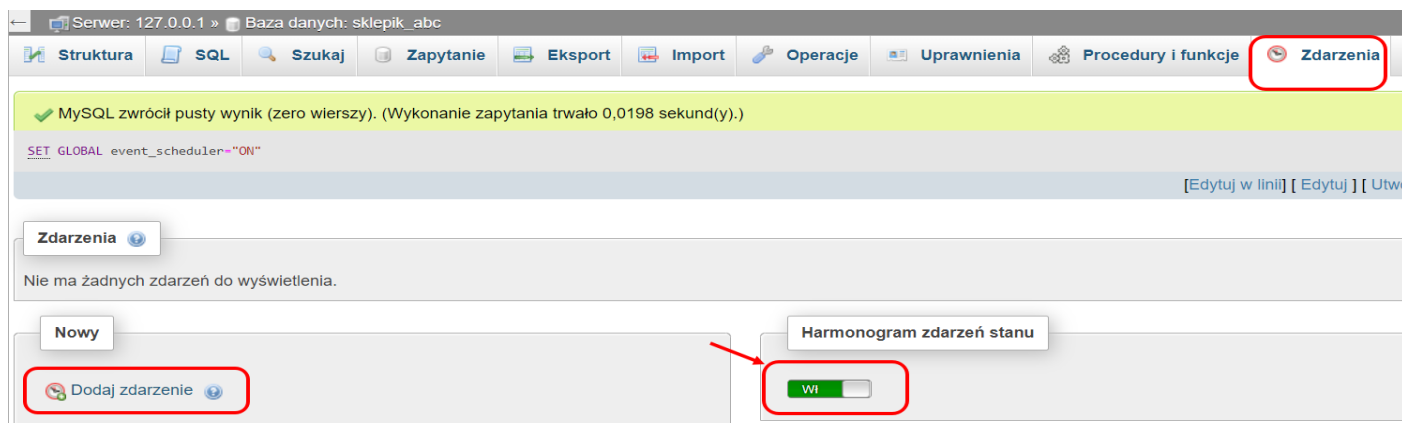
- AFTER DELETE – wykonanie wyzwalacza po operacji usunięcia rekordu
- AFTER INSERT – wykonanie wyzwalacza po dodaniu rekordu
- AFTER UPDATE – wykonanie wyzwalacza po zmodyfikowaniu rekordu
- BEFORE DELETE – wykonanie wyzwalacza przed operacji usunięcia rekordu
- BEFORE INSERT – wykonanie wyzwalacza przed dodaniem rekordu
- BEFORE UPDATE – wykonanie wyzwalacza przed zmodyfikowaniem rekordu

Przykład: na bazie sklepik_abc

Po wstawieniu jakiegoś rekordu do tabeli kategorie, zostanie automatycznie wstawiony rekord do producenta test_wyzwalacza

```
CREATE TRIGGER `wyzwalacz1` AFTER INSERT ON `kategorie` FOR EACH ROW INSERT INTO `producent` (`id_producenta`, `nazwa_producenta`) VALUES (NULL, 'test_wyzwalacza');
```

Dodawanie graficznie zdarzeń w xamppie



Przykład: przez minutę co 15 sekund będzie się wstawiało do tabeli kategorie rekord test

Dodaj zdarzenie ✕

Szczegóły

Nazwa zdarzenia	dodaj_kategorie		
Status	DISABLED ▼		
Typ zdarzenia	RECURRING ▼		
Wykonaj co	15	SECOND ▼	
Pocz	2021-01-03 13:55:00.000000 📅		
Koniec	2021-01-03 13:56:00.000000 📅		
	1 INSERT INTO kategorie VALUES (null, 'test')		

```
CREATE EVENT `dodaj_kategorie` ON SCHEDULE EVERY 15 SECOND STARTS '2021-01-03 13:55:00.000000' ENDS '2021-01-03 13:56:00.000000' ON COMPLETION NOT PRESERVE ENABLED DO INSERT INTO kategorie VALUES (null, 'test');
```

Należy pamiętać o przestawieniu statusu na WŁĄCZONY –ENABLED.

Nazwa	Status	Działanie	Typ
☐ dodaj_kategorie	ENABLED	Edytuj Eksport Usuń	RECURRING

Pokazane wszystkie kategorie:

SELECT * FROM `kategorie`

☐ Pokaż wszystko | Liczba wierszy: | Filtrowanie wierszy:

Opcje

	id_kategoria	nazwa
☐ Edytuj Kopiuj Usuń	1	AGD
☐ Edytuj Kopiuj Usuń	2	sprzęt informatyczny
☐ Edytuj Kopiuj Usuń	3	spożywcze
☐ Edytuj Kopiuj Usuń	4	chemiczne
☐ Edytuj Kopiuj Usuń	5	sprzęt sportowy
☐ Edytuj Kopiuj Usuń	6	instrumenty muzyczne
☐ Edytuj Kopiuj Usuń	7	RTV
☐ Edytuj Kopiuj Usuń	8	samochody
☐ Edytuj Kopiuj Usuń	9	zabawki
☐ Edytuj Kopiuj Usuń	10	obuwie
☐ Edytuj Kopiuj Usuń	11	odzież

☐ Zaznacz wszystko | Z zaznaczonymi: Edytuj Kopiuj

Rysunek 2. Kategorie przed wykonaniem zdarzenia

	id_kategoria	nazwa
☐ Edytuj Kopiuj Usuń	1	AGD
☐ Edytuj Kopiuj Usuń	2	sprzęt informatyczny
☐ Edytuj Kopiuj Usuń	3	spożywcze
☐ Edytuj Kopiuj Usuń	4	chemiczne
☐ Edytuj Kopiuj Usuń	5	sprzęt sportowy
☐ Edytuj Kopiuj Usuń	6	instrumenty muzyczne
☐ Edytuj Kopiuj Usuń	7	RTV
☐ Edytuj Kopiuj Usuń	8	samochody
☐ Edytuj Kopiuj Usuń	9	zabawki
☐ Edytuj Kopiuj Usuń	10	obuwie
☐ Edytuj Kopiuj Usuń	11	odzież
☐ Edytuj Kopiuj Usuń	12	test
☐ Edytuj Kopiuj Usuń	13	test
☐ Edytuj Kopiuj Usuń	14	test
☐ Edytuj Kopiuj Usuń	15	test
☐ Edytuj Kopiuj Usuń	16	test

Rysunek 1. Kategorie po wywołaniu zdarzenia

Transakcje- zbiór operacji na bazie danych stanowiących logiczną całość – albo żadną z nich.

Ćwiczenie przypominające o transakcjach. Można wykonać na dowolnej bazie. Przetestować, zrozumieć zasadę działania, a następnie usunąć tę tabelę konta.

Tworzenie tabeli:

```
CREATE TABLE konta
(id INTEGER PRIMARY KEY,
id_osoby INTEGER NOT NULL,
saldo DECIMAL(12,2) NOT NULL);
```

Wstawianie danych:

```
INSERT INTO konta VALUES (1,1,200.00);
INSERT INTO konta VALUES (2,2,300.00);

UPDATE `konta` SET saldo=saldo-40 WHERE `id`=1;
UPDATE `konta` SET saldo=saldo+40 WHERE `id`=2;

START TRANSACTION;

UPDATE `konta` SET saldo=saldo-40 WHERE `id`=1;
UPDATE `konta` SET saldo=saldo+40 WHERE `id`=2;

COMMIT;

SELECT * FROM `konta`;
```

ODWOLANIE TRANSAKCJI:

```
START TRANSACTION;

UPDATE `konta` SET saldo=saldo-40 WHERE `id`=1;
UPDATE `konta` SET saldo=saldo+40 WHERE `id`=2;

ROLLBACK;
```

Tworzenie kopii zapasowej bazy danych MySQL

Bazy danych zazwyczaj są tworzone aby przechowywać i przetwarzać jakieś ważne dane. Dlatego bardzo ważnym jest tworzenie kopii zapasowych pozwalające na zabezpieczenie informacji na wypadek zniszczenia danych lub awarii serwera.

Polecenie `mysqldump` służy do tworzenia plików tekstowych „dumps” baz danych zarządzanych MySQL. W tych plikach znajdują się wszystkie polecenia SQL, potrzebne do odtworzenia bazy danych od podstaw.

Otwieramy xampp-a, →shell→otwieramy okienko wiersza poleceń. Chcemy utworzyć kopię bazy sklepik_abc. Musimy napisać polecenie:

```
mysqldump sklepik_abc -u root -p > kopia_sklepikabc.sql
```

i wcisnąć ENTER. Dalej musimy podać hasło (u nas jest puste, więc ENTER i to wszystko).

```
kasia@DESKTOP: c:\xampp
# mysqldump sklepik_abc -u root -p > kopia_sklepikabc.sql
Enter password:

kasia@DESKTOP | c:\xampp
```

Znajdź w katalogu głównym Xamppa plik kopia_sklepikabc.sql

Ten komputer > Dysk lokalny (C:) > xampp		
Nazwa		Data modyfikacji
kopia_sklepikabc.sql		03.01.2021 20:59

Przywracanie kopii zapasowej

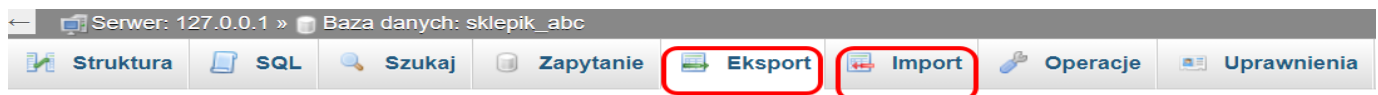
Otwieramy xampp-a, →shell→otwieramy okienko wiersza poleceń. Chcemy odtworzyć kopię bazy kopia_sklepikabc.sql. Musimy napisać polecenie:

```
mysqldump sklepik_abc -u root -p < kopia_sklepikabc.sql
```

i wcisnąć ENTER. Dalej musimy podać hasło (u nas jest puste, więc ENTER i to wszystko). Chwilę czekamy, aż zostaną wykonane wszystkie zapytania zapisane w pliku zrzutu.

```
# mysqldump sklepik_abc -u root -p < kopia_sklepikabc.sql
Enter password:
```

Oczywiście tworzenie kopii i przywracanie bazy można wykonać graficznie z zakładki Import /Eksport.



Eksportowanie tabeli z bazy "sklepik_abc"

Eksportuj szablony:

Nowy szablon:

Istniejący szablony:

Szablon:

Zadanie do przećwiczenia.

1. Wykonaj kopie zapasowe 2 swoich baz danych, a następnie dokonaj jakiś zmian w tych bazach, np. usuń rekordy.
2. Przywróć kopie zapasowe tych baz z punktu wyżej.

Wyjaśnienie:

W **MySQL** określenie, że funkcja jest **deterministyczna (DETERMINISTIC)**, oznacza:

Dla tych samych danych wejściowych funkcja zawsze zwróci dokładnie ten sam wynik.

Funkcja niedeterministyczna

Funkcja **niedeterministyczna** może zwrócić różne wyniki nawet dla tych samych danych.

Przykład:

```
CREATE FUNCTION losowa()
RETURNS DOUBLE
BEGIN
    RETURN RAND();
END
```

Tutaj wynik będzie inny:

```
SELECT losowa(); → 0.1234
SELECT losowa(); → 0.8765
```

Przypomnienie typów danych:

orównanie z innymi typami liczbowymi

Typ	Co przechowuje	Przykład
INT	liczby całkowite	10
FLOAT	liczby z przecinkiem (mniejsza dokładność)	3.14
DOUBLE	liczby z przecinkiem (większa dokładność)	3.141592653
DECIMAL	dokładne liczby np. pieniądze	10.50

DOUBLE

DOUBLE to liczba **zmiennoprzecinkowa (floating-point)**.

Oznacza to, że liczba jest zapisywana **przybliżenie**, a nie dokładnie.

Cechy:

- zapis w **64 bitach**
- bardzo duży zakres liczb
- szybkie obliczenia
- może pojawić się **mały błąd zaokrąglenia**

Przykład

```
SELECT 0.1 + 0.2;
```

może dać wynik:

```
0.30000000000000004
```

czyli nie idealnie 0.3.

2. DECIMAL

DECIMAL przechowuje liczby **dokładnie**, bez błędów zaokrągleń.

Jest to tzw. **liczba stałoprzecinkowa (fixed-point)**.

Cechy:

- bardzo **dokładny zapis**
- używany do **pieniędzy**
- wolniejszy w obliczeniach
- definiujesz ile cyfr i ile po przecinku

Składnia

```
DECIMAL (M, D)
```

gdzie:

- M – liczba wszystkich cyfr
- D – liczba cyfr po przecinku

Przykład

```
DECIMAL (10, 2)
```

może zapisać np.:

```
12345678.90
```

◇ Porównanie

Cecha	DOUBLE	DECIMAL
Typ liczby	zmiennoprzecinkowa	stałoprzecinkowa
Dokładność	przybliżona	dokładna
Szybkość	szybszy	wolniejszy
Zastosowanie	obliczenia matematyczne, naukowe pieniądze, ceny	

Przykłady funkcji:

Nazwa procedury

oblicz_wiek

Typ

Change to PROCEDURE

FUNCTION

Parametry

⬆

Nazwa

data_urodzenia

Typ

DATE

▼

Add parameter

Usuń ostatni parametr

Zwracany typ

INT

▼

Powrót długości/wartości

11

Powrót opcji

Charset

▼

▼

Definition

```

BEGIN
    RETURN TIMEDIFF(YEAR,
data_urodzenia, CURDATE());
END

```

```

SELECT imie, oblicz_wiek(data_urodzenia) AS wiek
FROM osoby;

```

Strona 11 z 11