

# 

# Wydział Studiów Strategicznych i Technicznych

## Kierunek: Informatyka, rok II, semestr III (2021/2022)

**BAZ DANYCH (SQL)**

Prowadzący: dr hab. Filip Rudziński

**Autor sprawozdania:**

Magdalena Szafrańska, nr albumu: 18345

# Spis treści

[**Wstęp**](#_eqrekgah3ij1) **3**

[Wykonanie projektu bazy danych](#_7hcmaw71zv34) 3

[Uwaga](#_sgn8zsxslmup) 3

[**Cel projektu**](#_zacsedhsz1lo) **3**

[Użyte narzędzia](#_mbka1ch5asup) 4

[Projekt bazy danych z użyciem diagramów ERD](#_ej8b4v9xhn16) 4

[Wstępna struktura bazodanowa](#_to8mz253sl1n) 4

[Uruchomienie serwera bazodanowego w narzędziu XAMPP](#_mv6uf5oihu5w) 5

[Baza danych w serwerze lokalnym XAMPP](#_303djdiksmw1) 6

[Uruchomienie narzędzia phpMyAdmin](#_yxfxo898mgte) 6

[Utworzenie bazy danych](#_6gkk9y28yudv) 6

[Generowanie podstawowego kodu w MySQL](#_iovjc1i7v9p8) 6

[Stworzenie encji](#_mrpn2rfxvfkd) 6

[Stworzenie relacji pomiędzy encjami](#_nbx5ryg93ecm) 8

[Diagram relacji pomiędzy tabelami](#_d0qpld588au1) 9

[**Wprowadzenie danych do bazy**](#_qe6vj9s9x5cs) **9**

[**Przykładowe zapytania SQL**](#_5acv9ht5qe8v) 11

[Influencerzy posortowani malejąco wg liczby followersów (ORDER BY)](#_3g2n9iwgsed2) 11

[Akcje wykonane przez influencerów w listopadzie (BETWEEN)](#_v8ttkt4k08mn) 11

[Wszystkie produkty z “Reductor” w nazwie (LIKE)](#_l1s96xugql1f) 12

[Jaka wysyłka, o jakiej wartości i kiedy trafiła do influencerów (AS)](#_u0cuql8xp1af) 12

[Funkcje wbudowane SQL](#_2djykdrtpro9) 13

[ilość wszystkich produktów (COUNT)](#_mnfw0mq9rww) 13

[średni koszt możliwych do wykonania przez influencera akcji](#_5691b5b555n) 13

[Wysyłki konkretnych produktów do influencerów](#_5n286gkdcxcw) 14

[wszyscy influencerzy, którym wysłano (LIKE + AND)](#_oe7wltwb9ens) 14

[ograniczenie ilości do 5 (LIMIT)](#_13kvimvvgfnb) 14

[Wszystkie wysyłki do określonego influencera](#_wkmgzv7s80mb) 15

[Aktualizacja imienia influencera (UPDATE)](#_6ze0xt5x8y4h) 15

[Influencerzy spoza Polski (NOT IN)](#_g6or5gr46ix5) 16

[Akcje wykonane przez influencerów](#_ipo4kunempms) 17

[influencerzy, którzy nie wykonali żadnej akcji (LEFT JOIN)](#_gu7fjkq0bfko) 17

[influencerzy, którzy wykonali jakąkolwiek akcję (INNER JOIN)](#_uk50ri5ozb5k) 17

[Wyszukanie produktów nigdy nie wysłanych (NOT EXISTS)](#_b7cua7crxzum) 18

[**Widoki**](#_ujcorvo309g7) **19**

[**Funkcje**](#_wi001xald0lg) **20**

[Połączenie imienia i nazwiska](#_tmg5vy42frs4) 20

[Etykieta dla influencera wg followersów](#_m8hahwbi346q) 20

[Wartość wysyłki w kursie euro](#_9k47gwdpyxyv) 22

[**Procedury**](#_tgyurwknfqyq) **23**

[Wyświetlanie danych o influencerach](#_ri5ltmmkw6xw) 24

[Wyświetlanie działań wszystkich influencerów](#_2aruiyhshj8a) 25

[Wyświetlanie działań konkretnego influencera](#_mvysrtca1yhg) 26

[Wyświetlanie wynagrodzenia dla influencera w danym miesiącu](#_657068kzhyvb) 28

[Zmniejszenie ilość towarów w magazynie](#_pczynh6ijhcp) 29

[**Obsługa wyjątków**](#_q760znf3q2jq) **31**

[**Wyzwalacze**](#_5izj25k58261) **32**

[Wywołanie procedury zmniejszania ilości magazynu](#_fxlrp64isux4) 33

[Dodanie nowego rekordu](#_wjqa6dy67va0) 35

[**Przydzielanie uprawnień użytkownikom**](#_bmh2cp8uo61n) 37

[Tworzenie nowego użytkownika z uprawnieniami READONLY](#_z833by7snb01) 38

[Tworzenie nowego użytkownika z uprawnieniami ‘full access’](#_pit35glzf7em) 41

[**Pełny skrypt SQL**](#_pocaavekfrbr) **44**

# 

# Wstęp

Podstawowym etapem powstawania bazy danych jest tworzenie jej projektu. Od decyzji podjętych na etapie projektowania zależeć będzie jakość i użyteczność stworzonej bazy danych. Baza danych, podobnie jak większość projektów komputerowych, jest modelem wycinka świata rzeczywistego, utworzonym tak, aby był możliwy do zapamiętania przez maszynę cyfrową i zawierał optymalną ilość informacji do zastosowania, jakiemu będzie służył.

Przy modelowaniu baz danych możemy posłużyć się notacją graficzną modelowania danych – diagramem związków encji ERD (ang. Entity-Relationship Diagram). Jest to model sieciowy opisujący na wysokim poziomie abstrakcji dane, które są przechowywane w systemie.

Każda nowo utworzona baza danych wymaga konsekwentnego etapowego działania. Cały proces projektowania bazy danych możemy podzielić na kilka etapów:

* planowanie bazy danych,
  + określenie występujących zbiorów encji,
  + określenie atrybutów przypisanych do poszczególnych encji
  + określenie dziedziny poszczególnych atrybutów
* tworzenie modelu konceptualnego (np. diagramu ERD),
* transformacja modelu konceptualnego na model relacyjny,
* proces normalizacji bazy danych,
* wybór struktur i określenie zasad dostępu do bazy danych.

## Wykonanie projektu bazy danych

Zaprojektowałam bazę danych do serwisu monitorującego współpracę firmy z influencerami. Model związków encji zawiera 8 encji. Zaprojektowaną bazę planuję w przyszłości rozszerzyć i realnie wykorzystać w mojej pracy na co dzień, gdyż takiego właśnie narzędzia monitorującego brakuje w mojej obecnej firmie.

## Uwaga

Do pracy użyłam bazy danych tworzonej przeze mnie w tym samym czasie na laboratoriach z przedmiotu “Bazy danych (SQL)” pod przewodnictwem mgr. Tomasza Marka. W trakcie pracy nad pierwotną bazą, potrzeby, jakie niniejsza baza miała spełnić, odbiegały nieco od wersji pierwotnej przeznaczonej na laboratoria. W celu dokończenia projektu zaliczeniowego w obrębie konwersatoriów postanowiłam skopiować zawartość do nowej bazy pracując od tego momentu jedynie na niej. Nowa bazę nazwałam “db\_influencers\_konw”. Z tego właśnie powodu w początkowych poleceniach i screenach pojawia się nazwa bazy “db\_influencers”, a w późniejszych “db\_influencers\_konw”.

# Cel projektu

Świat cyfrowy daje coraz więcej możliwości do współpracy online. Firmy zawierają porozumienia z osobami obecnymi w social mediach (tzw. influencerami) wynagradzając ich za promowanie produktów marki, wysyłając im swoje produkty do testowania czy choćby nadając im specjalne kody rabatowe na zniżkę w sklepie internetowym.

Moim celem jest przygotowanie modelu bazy danych, który będzie umożliwiał m.in.:

* dodawanie i usuwanie influencerów,
* gromadził informację jakie działania marketingowe na rzecz marki wykonał dany influencer,
* jakie wynagrodzenie marka powinna wypłacić influencerowi za określone działania,
* kiedy i jakie produkty zostały wysłane do influencera,
* użycia jego kodu zniżkowego,
* przeszukanie osób (influencerów) generujących dla marki największe zyski.

## Użyte narzędzia

* język zapytań SQL
* MySQL – ogólnodostępny system zarządzania relacyjnymi bazami danych, który pomaga użytkownikom przechowywać, uporządkowywać i później pobierać dane
* lokalny serwer XAMPP w wersji v3.3.0
* silnik bazy danych MariaDB w wersji 10.4.21
* phpMyAdmin Version information: 5.1.1 - narzędzie służące do przetwarzania informacji znajdujących się w bazie danych i łatwego zarządzania bazą
* edytor GenMyModel do projektowanie baz danych (<https://www.genmymodel.com/>)

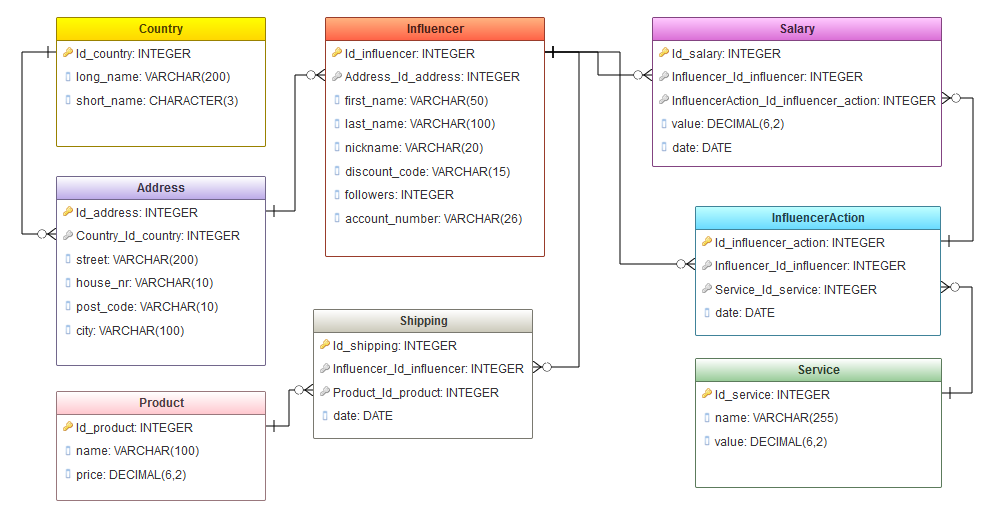
## Projekt bazy danych z użyciem diagramów ERD

Aby wstępnie zobrazować budowę bazy danych, opracowałam diagram związków encji, który będzie jednoznacznie i przejrzyście przedstawiał wymagania firmy w zakresie przetwarzanych przez nią danych oraz umożliwiał zbudowanie na jego podstawie relacyjnej bazy danych.

Do wykonania diagramu ERD użyłam edytora GenMyModel (<https://www.genmymodel.com/>) umożliwiającego projektowanie baz danych online na poziomie tabel i odniesień. Encje przedstawione są za pomocą prostokątów zawierających listę atrybutów. Klucze główne oznaczone są przez żółtą ikonę klucza.

Po wprowadzeniu relacji pomiędzy tabelami zostały dodane klucze obce. Poniżej diagram po wprowadzeniu relacji wraz z opcjonalnością związku oraz pokazaniem rodzaju relacji.

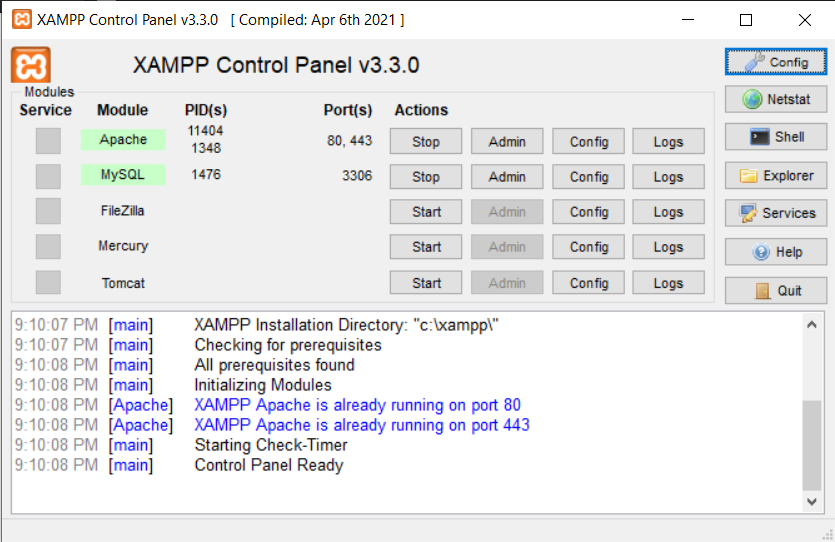
### Wstępna struktura bazodanowa



Tak przygotowany diagram ERD pozwala na późniejszą weryfikację i optymalizację bazy danych, a także stanowi podstawową dokumentację projektowanej bazy danych.

### Uruchomienie serwera bazodanowego w narzędziu XAMPP

W panelu kontrolnym pakietu XAMPP uruchamiam usługę “Apache” oraz “MySQL”. Po upewnieniu się, że serwer jest włączony wraz z usługą “MySQL”, zamykam panel bo on i tak pozostanie działający w tle. Następnie przechodzę do pakietu phpMyAdmin.



## Baza danych w serwerze lokalnym XAMPP

### Uruchomienie narzędzia phpMyAdmin

Do wykonania niniejszego projektu użyłam zainstalowanego na moim komputerze serwera lokalnego. XAMPP symuluje właśnie taki lokalny serwer dzięki specjalnemu adresowi sieciowemu w mojej karcie sieciowej: 127.0.0.1 (localhost). Wszystkie operacje odbywać się będą lokalnie na moim dysku, bez żadnych opóźnień.

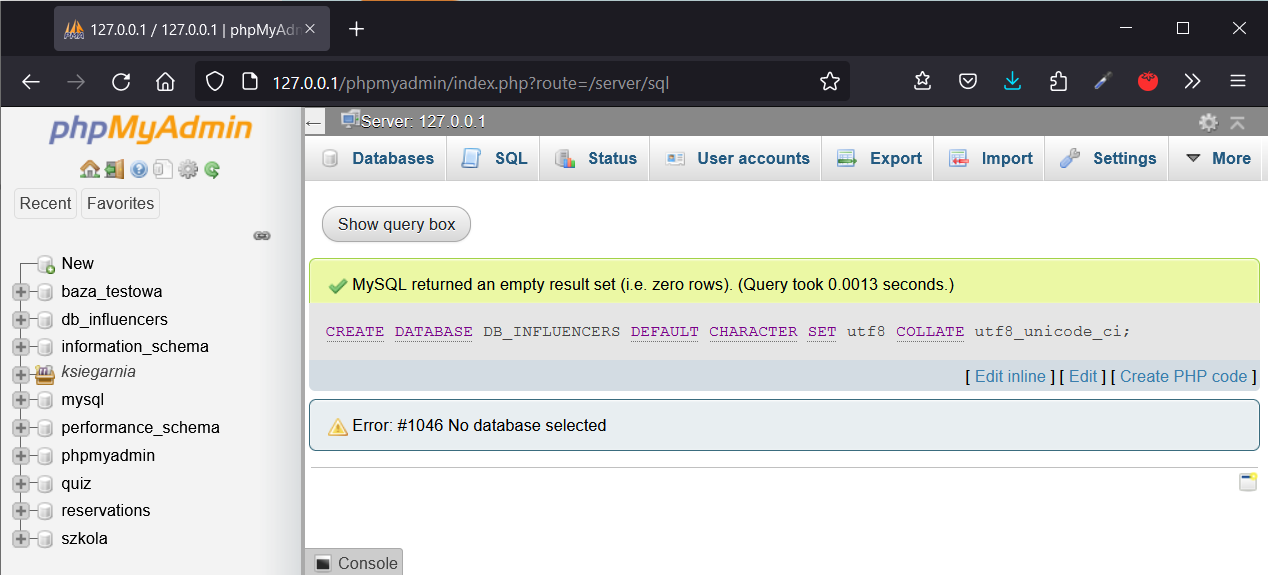
Taki lokalny serwer posłuży mi idealnie do nauki treści z wykładów i wykonania projektu zaliczeniowego. Nic nie stoi na przeszkodzie aby później gotowy serwis przenieść z mojego lokalnego serwera na serwer działający w Internecie i dostępny już dla wszystkich internautów.

### Utworzenie bazy danych

W narzędziu phpMyAdmin utworzyłam nową bazę danych o nazwie DB\_Influencers. Aby zapewnić poprawność wyświetlania polskich znaków zastosowałam dodatkowe polecenia SQL:

| CREATE DATABASE DB\_Influencers DEFAULT CHARACTER SET utf8 COLLATE utf8\_unicode\_ci; |
| --- |

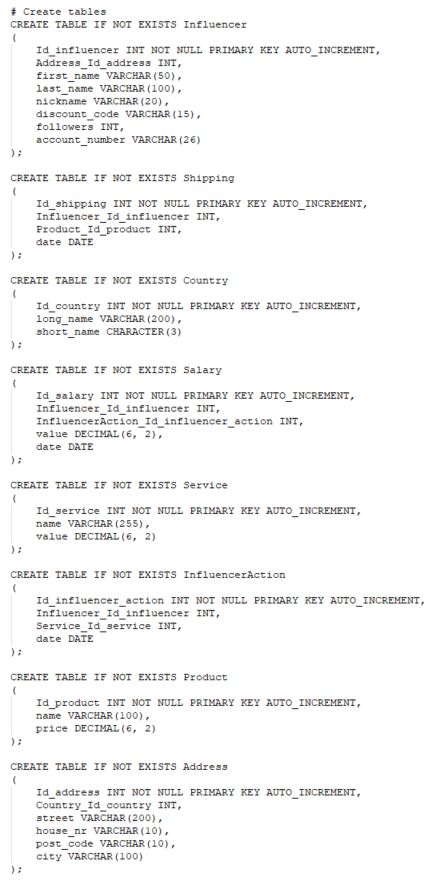
Baza danych została prawidłowo utworzona.



### Generowanie podstawowego kodu w MySQL

#### Stworzenie encji

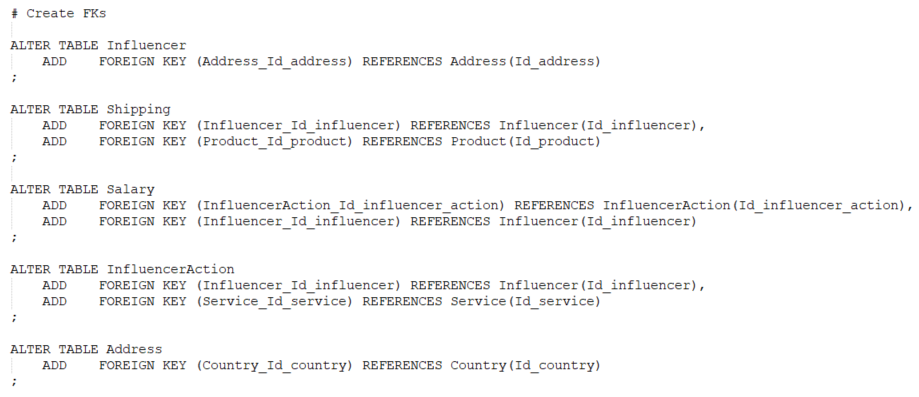
Skrypt wygenerowany w genMyModel.com podzieliłam na dwie części. Najpierw w oknie SQL wykonałam skrypt dotyczący tworzenia tabel. Kod przedstawiam poniżej. Po zatwierdzeniu tabele zostały pomyślnie utworzone, co również przedstawia poniższy zrzut ekranu.

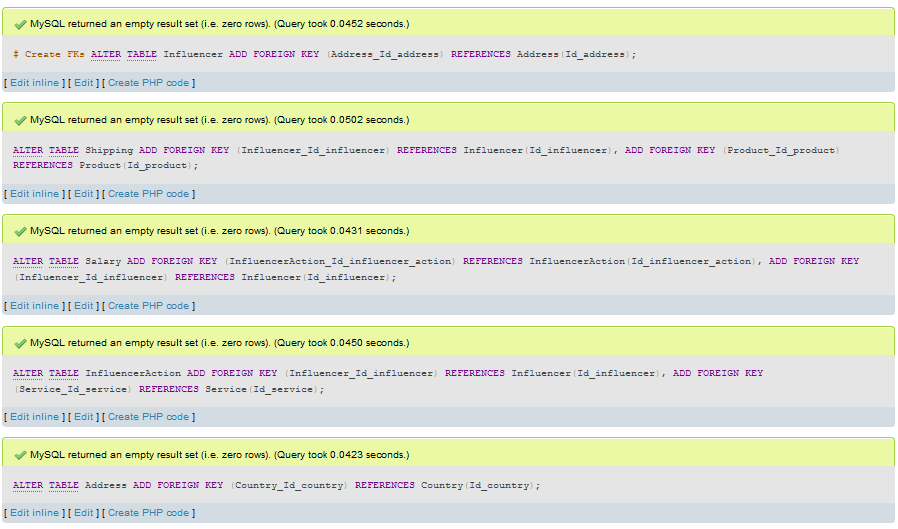




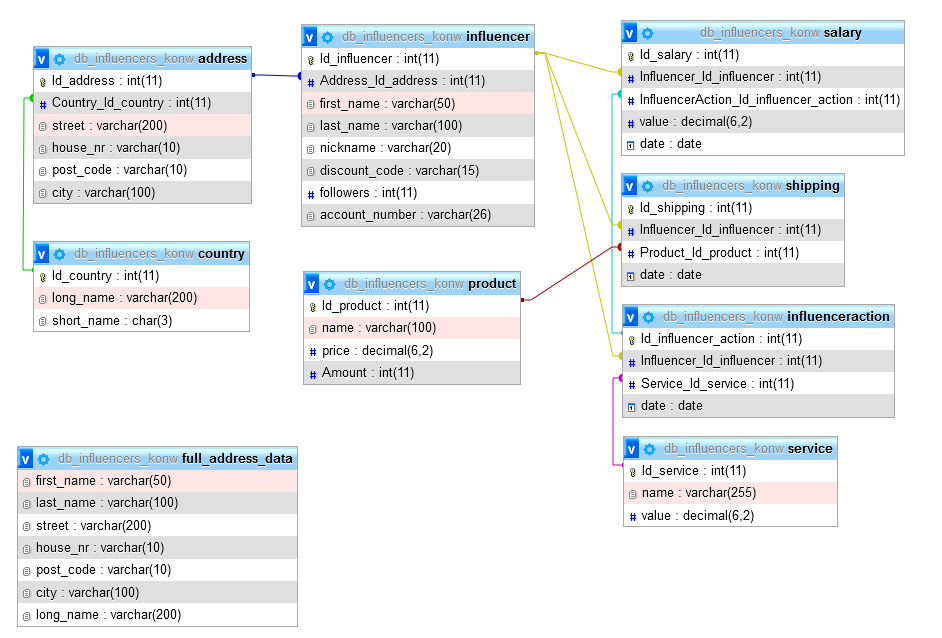
#### Stworzenie relacji pomiędzy encjami

Wprowadziłam skrypt odpowiadający za utworzenie relacji pomiędzy tabelami. Kod poniżej. Po zatwierdzeniu relacje pomiędzy encjami zostały pomyślnie utworzone, co również przedstawia poniższy zrzut ekranu.



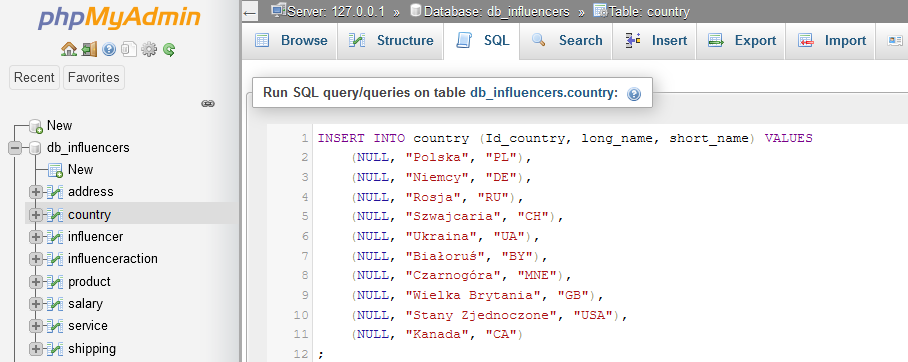


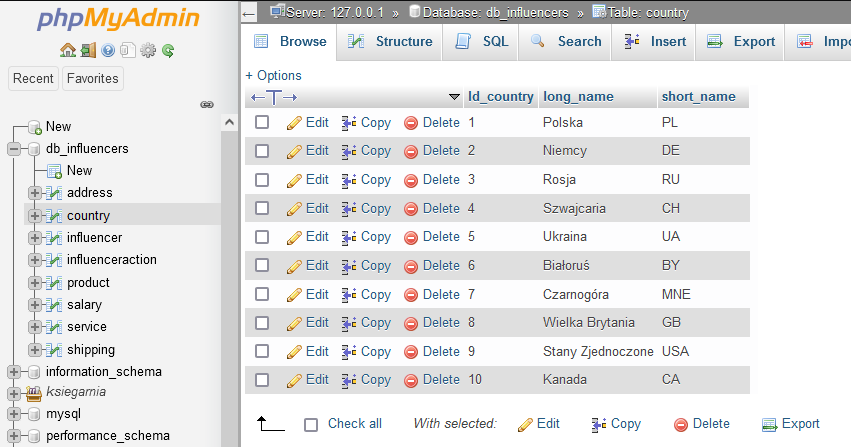
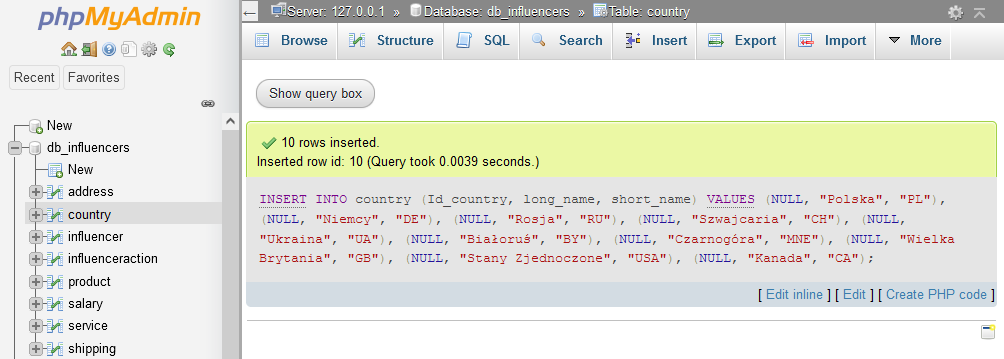
#### Diagram relacji pomiędzy tabelami



# Wprowadzenie danych do bazy

Używając języka SQL uzupełniłam zaprojektowane tabele danymi. Do każdej z nich wprowadzam co najmniej po 10 rekordów. Poniżej przykładowa procedura uzupełniania tabeli “Country”. Dla pozostałych tabel proces był analogiczny więc zamieszczę jedynie skrypt SQL dla każdej z nich. Pełny kod znajduje się na końcu sprawozdania w zakładce [Pełny skrypt SQL](#_811axzokj3eu).





## Przykładowe zapytania SQL

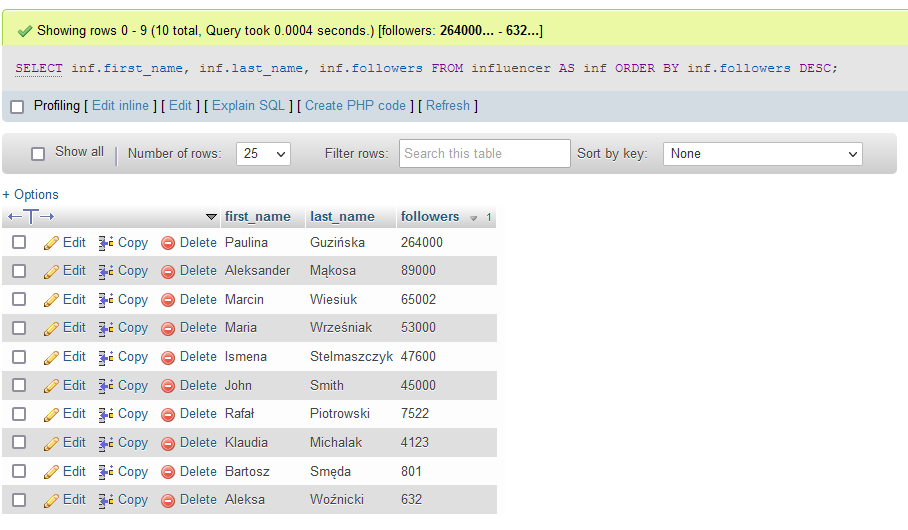
Używając języka SQL stworzyłam ponad 10 dowolnych zapytań do zaprojektowanej bazy. Podczas tworzenia zapytań złożonych dostarczyłam listę wszystkich relacji, które zachodzą pomiędzy używanymi w tym zapytaniu tabelami. Tę listę umieszczałam po klauzuli WHERE.

W przykładach użyłam zarówno zapytań złożonych jak i skorelowanych.

### Influencerzy posortowani malejąco wg liczby followersów (ORDER BY)

**POLECENIE SQL:**

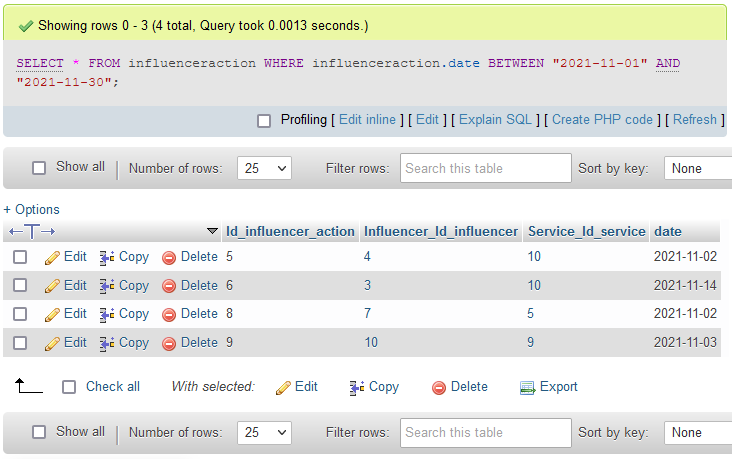
| SELECT inf.first\_name, inf.last\_name, inf.followers FROM influencer AS inf  ORDER BY inf.followers DESC; |
| --- |



### Akcje wykonane przez influencerów w listopadzie (BETWEEN)

**POLECENIE SQL:**

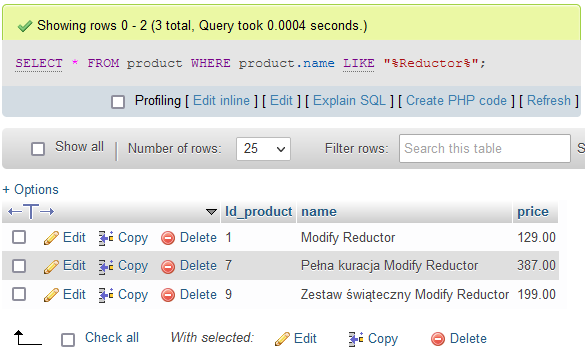
| SELECT \* FROM influenceraction WHERE influenceraction.date  BETWEEN "2021-11-01" AND "2021-11-30"; |
| --- |



### Wszystkie produkty z “Reductor” w nazwie (LIKE)

**POLECENIE SQL:**

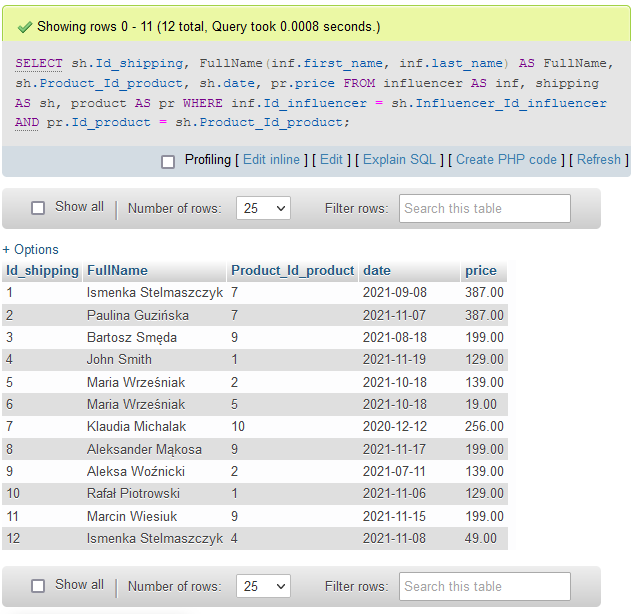
| SELECT \* FROM product WHERE product.name LIKE "%Reductor%"; |
| --- |



### Jaka wysyłka, o jakiej wartości i kiedy trafiła do influencerów (AS)

**POLECENIE SQL:**

| SELECT sh.Id\_shipping, FullName(inf.first\_name, inf.last\_name)  AS FullName, sh.Product\_Id\_product, sh.date, pr.price  FROM influencer AS inf, shipping AS sh, product AS pr  WHERE inf.Id\_influencer = sh.Influencer\_Id\_influencer  AND pr.Id\_product = sh.Product\_Id\_product; |
| --- |

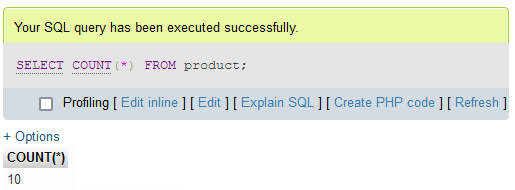


### Funkcje wbudowane SQL

#### ilość wszystkich produktów (COUNT)

**POLECENIE SQL:**

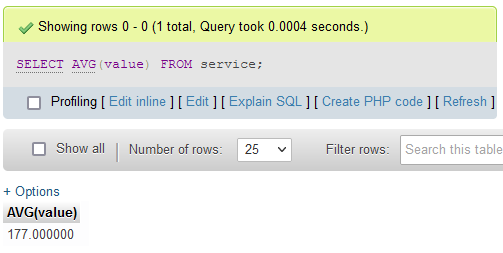
| SELECT COUNT(\*) FROM product; |
| --- |

****

#### średni koszt możliwych do wykonania przez influencera akcji

**POLECENIE SQL:**

| SELECT AVG(value) FROM service; |
| --- |

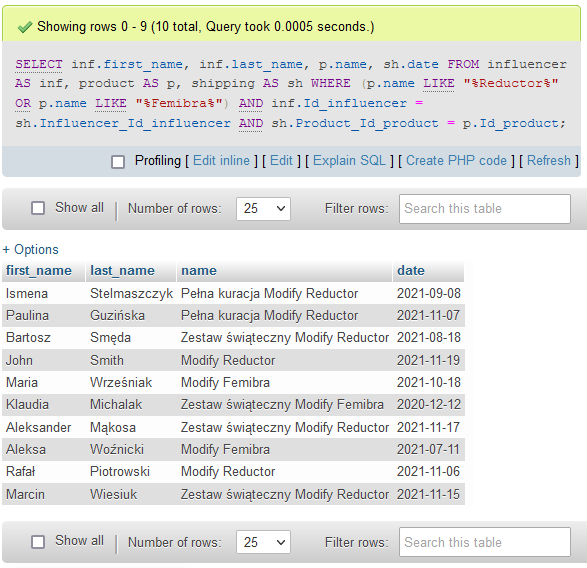


### Wysyłki konkretnych produktów do influencerów

#### wszyscy influencerzy, którym wysłano (LIKE + AND)

**POLECENIE SQL:**

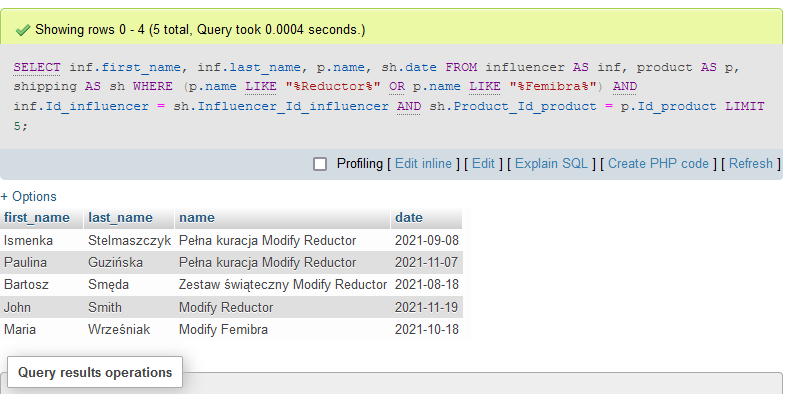
| SELECT inf.first\_name, inf.last\_name, p.name, sh.date  FROM influencer AS inf, product AS p, shipping AS sh  WHERE (p.name LIKE "%Reductor%" OR p.name LIKE "%Femibra%")  AND inf.Id\_influencer = sh.Influencer\_Id\_influencer  AND sh.Product\_Id\_product = p.Id\_product  LIMIT 5; |
| --- |



#### ograniczenie ilości do 5 (LIMIT)

**POLECENIE SQL:**

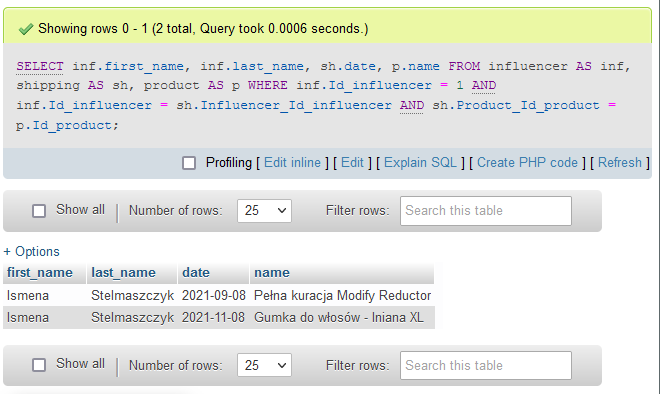
| SELECT inf.first\_name, inf.last\_name, p.name, sh.date  FROM influencer AS inf, product AS p, shipping AS sh  WHERE (p.name LIKE "%Reductor%" OR p.name LIKE "%Femibra%")  AND inf.Id\_influencer = sh.Influencer\_Id\_influencer  AND sh.Product\_Id\_product = p.Id\_product  LIMIT 5; |
| --- |



### Wszystkie wysyłki do określonego influencera

**POLECENIE SQL**:

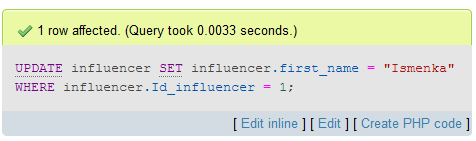
| SELECT inf.first\_name, inf.last\_name, sh.date, p.name  FROM influencer AS inf, shipping AS sh, product AS p  WHERE inf.Id\_influencer = 1  AND inf.Id\_influencer = sh.Influencer\_Id\_influencer  AND sh.Product\_Id\_product = p.Id\_product; |
| --- |



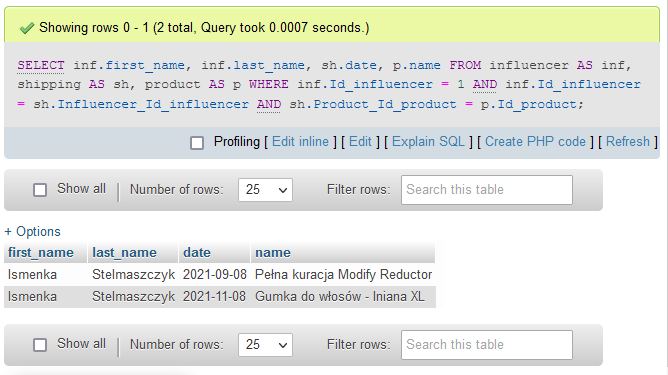
### Aktualizacja imienia influencera (UPDATE)

**POLECENIE SQL:**

| UPDATE influencer SET influencer.first\_name = "Ismenka"  WHERE influencer.Id\_influencer = 1; |
| --- |



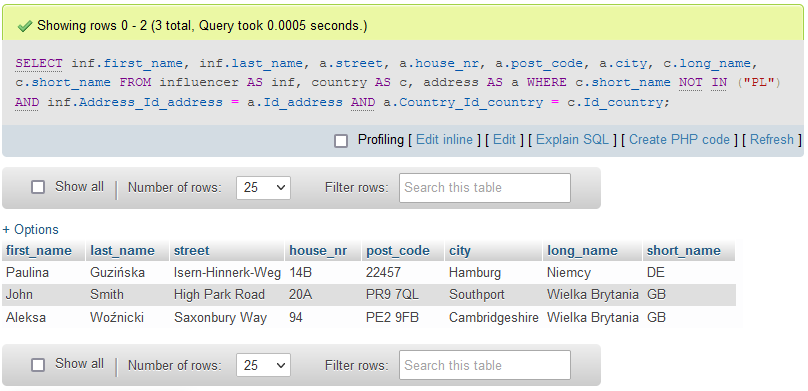
Wykonanie ponownie skryptu z poprzedniego punktu dla sprawdzenia.



### Influencerzy spoza Polski (NOT IN)

**POLECENIE SQL**:

| SELECT inf.first\_name, inf.last\_name, a.street, a.house\_nr, a.post\_code, a.city, c.long\_name, c.short\_name  FROM influencer AS inf, country AS c, address AS a  WHERE c.short\_name NOT IN ("PL")  AND inf.Address\_Id\_address = a.Id\_address  AND a.Country\_Id\_country = c.Id\_country; |
| --- |



### Akcje wykonane przez influencerów

#### influencerzy, którzy nie wykonali żadnej akcji (LEFT JOIN)

**POLECENIE SQL:**

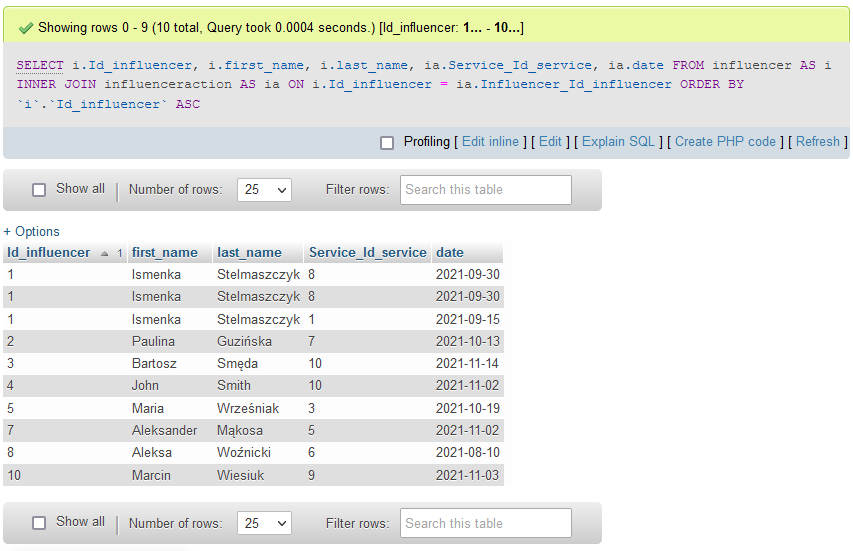
| SELECT i.Id\_influencer, i.first\_name, i.last\_name  FROM influencer AS i LEFT JOIN influenceraction AS ia  ON i.Id\_influencer = ia.Influencer\_Id\_influencer  WHERE ia.Influencer\_Id\_influencer IS NULL; |
| --- |



#### influencerzy, którzy wykonali jakąkolwiek akcję (INNER JOIN)

**POLECENIE SQL:**

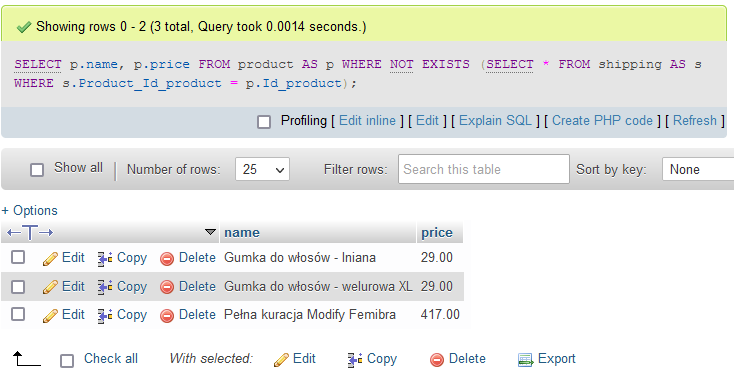
| SELECT \* FROM influencer AS i  INNER JOIN influenceraction AS ia  ON i.Id\_influencer = ia.Influencer\_Id\_influencer; |
| --- |



### Wyszukanie produktów nigdy nie wysłanych (NOT EXISTS)

**POLECENIE SQL:**

| SELECT p.name, p.price FROM product AS p WHERE NOT EXISTS  (SELECT \* FROM shipping AS s WHERE s.Product\_Id\_product = p.Id\_product); |
| --- |



## Widoki

Widoki (perspektywy) w języku SQL to wirtualne tabele tworzone na podstawie zapytań. Składają się z kolumn i wierszy pobranych z prawdziwych tabel. Pokazywane w widoku dane są zawsze aktualne, ponieważ widoki są tworzone w momencie wykonania zapytania. Widoki nie przechowują zapisanych w tabelach danych.

Aby zaprezentować tę funkcjonalność stworzyłam widok z odpowiednich kolumn oparty na następujących tabelach:

* tabela “influencer”
  + kolumna “first\_name”
  + kolumna “last\_name”
* tabela “address”
  + kolumna “street”
  + kolumna “house\_nr”
  + kolumna “post\_code”
  + kolumna “city”
* tabela “country”
  + kolumna “long\_name”

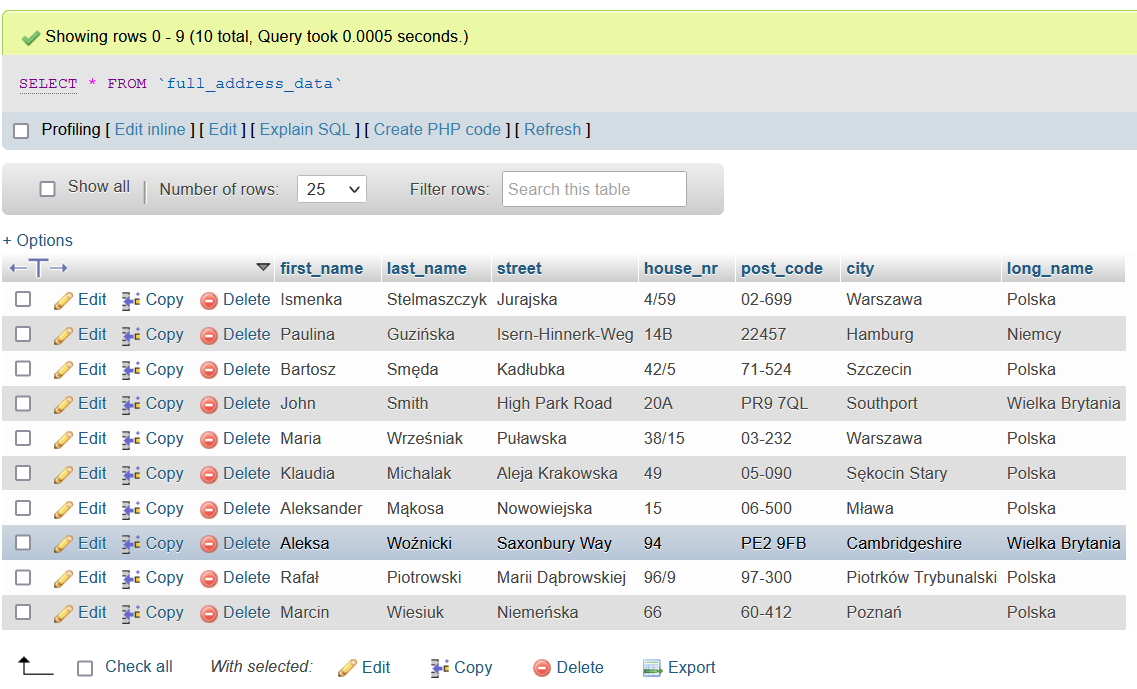
Kod do wygenerowanie widoku:

| SELECT i.first\_name AS first\_name, i.last\_name AS last\_name, a.street AS street,  a.house\_nr AS house\_nr, a.post\_code AS post\_code, a.city AS city,  c.long\_name AS long\_name  FROM ((db\_influencers\_konw.influencer i left join db\_influencers\_konw.address a on(i.Address\_Id\_address = a.Id\_address)) left join db\_influencers\_konw.country c on(a.Country\_Id\_country = c.Id\_country)) |
| --- |

Stworzony widok wygenerował tabelę jak na poniższym zrzucie ekranu.

**POLECENIE SQL:**

| SELECT \* FROM full\_address\_data; |
| --- |



## Funkcje

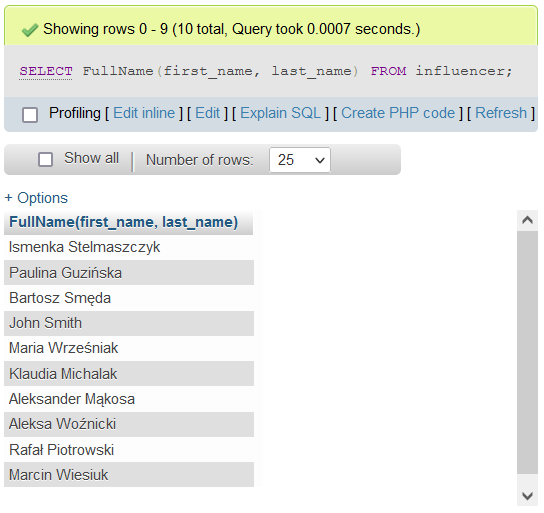
Funkcji używa się, aby tworzyć dedykowane rozwiązania. Funkcjom, tak jak procedurom, można przekazać pewną liczbę parametrów. Funkcja jednak nie tylko wykonuje pewne operacje, ale także zwraca obliczony na podstawie przekazanych parametrów wynik.

### Połączenie imienia i nazwiska

Funkcja wyświetlająca imię i nazwisko jako jedną frazę. Często używam tej funkcji w bazie więc zdecydowałam się na funkcję skracająca mi czas wywoływania imienia i nazwiska osobno za każdym razem.

**POLECENIE SQL:**

| CREATE FUNCTION FullName(first\_name VARCHAR(20), last\_name VARCHAR(30))  RETURNS VARCHAR(50)  DETERMINISTIC  RETURN CONCAT(first\_name, ' ', last\_name); |
| --- |



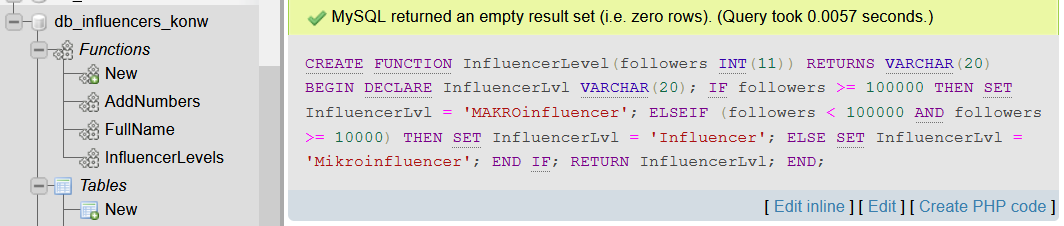
### Etykieta dla influencera wg followersów

Przypisanie influencerowi odpowiedniej “etykiety” w zależności od liczby jego followersów.

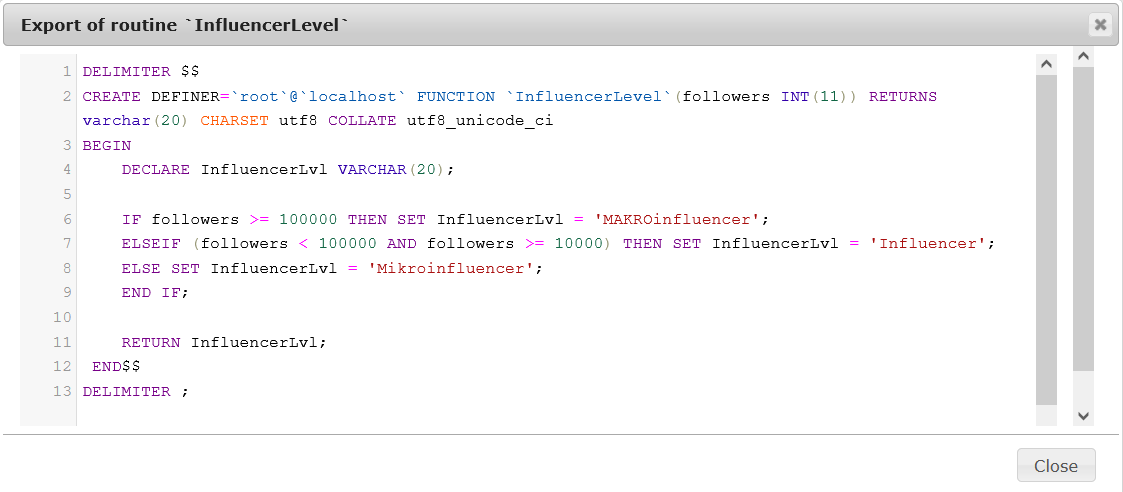
**POLECENIE SQL:**

| DELIMITER //  CREATE FUNCTION InfluencerLevel(followers INT(11))  RETURNS VARCHAR(20)  BEGIN  DECLARE InfluencerLvl VARCHAR(20);    IF followers >= 100000 THEN SET InfluencerLvl = 'MAKROinfluencer';  ELSEIF (followers < 100000 AND followers >= 10000) THEN SET InfluencerLvl = 'Influencer';  ELSE SET InfluencerLvl = 'Mikroinfluencer';  END IF;    RETURN InfluencerLvl;  END //  DELIMITER ; |
| --- |

Kod został zaimplementowany pomyślnie (screen poniżej).



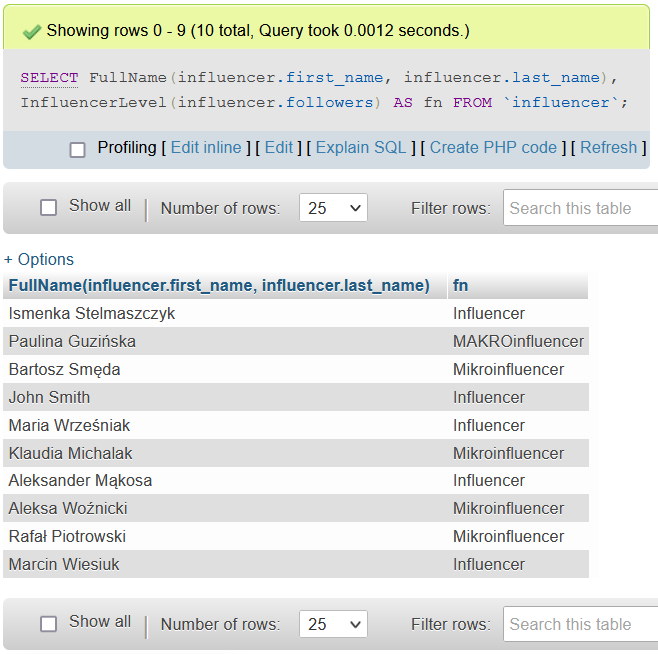
Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Połączenie dwóch powyższych funkcji pozwoliło mi wygenerować zapytanie złożone. Korzystając w nim z pierwszej funkcji wyświetliłam imiona i nazwiska wszystkich influencerów z tabeli Influencerzy. Druga funkcja natomiast przyporządkowała danego influencera w osobnej kolumnie *fn* do jednej z trzech grup liczności (w zależności od liczby followersów tegoż influencera).

**POLECENIE SQL:**

| SELECT FullName(influencer.first\_name, influencer.last\_name), InfluencerLevel(influencer.followers) AS fn FROM `influencer`; |
| --- |



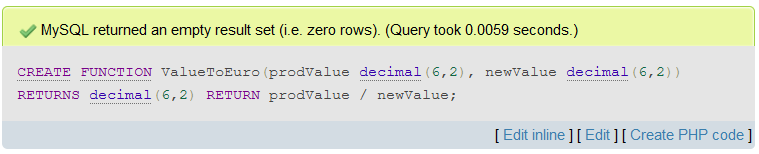
### Wartość wysyłki w kursie euro

W funkcji przekazuję wartość wysyłki w złotych i podaję na bieżąco kurs euro. W przyszłości planuję rozwinąć funkcjonalność i pobierać bieżący kurs euro z ogólnodostępnych źródeł.

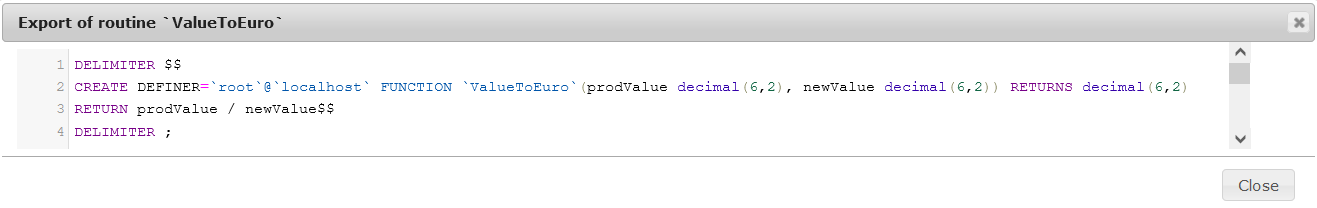
**POLECENIE SQL:**

| CREATE FUNCTION ValueToEuro(prodValue decimal(6,2), newValue decimal(6,2))  RETURNS decimal(6,2)  RETURN prodValue / newValue; |
| --- |

Kod został zaimplementowany pomyślnie (screen poniżej).



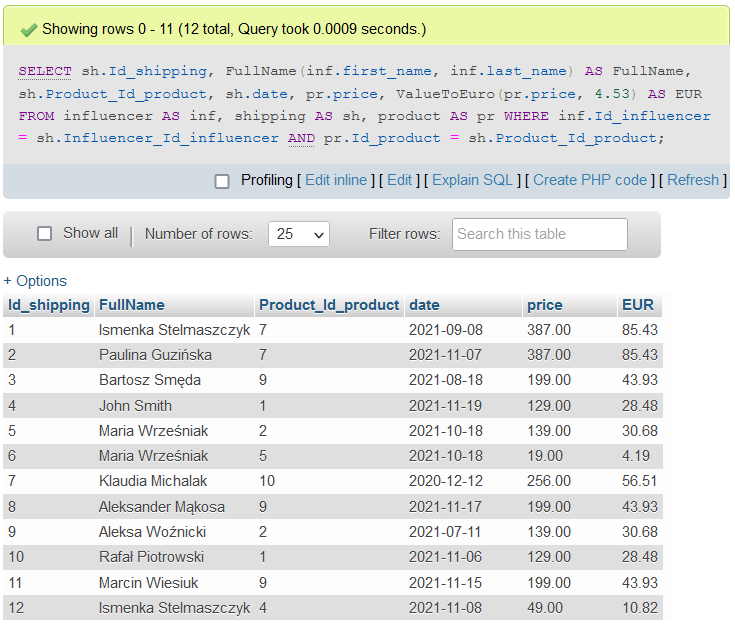
Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Połączenie pierwszej oraz niniejszej funkcji pozwoliło mi wygenerować zapytanie złożone. Wyświetliłam jaka wysyłka, o jakiej wartości i kiedy trafiła do danego influencera.

Korzystając z pierwszej funkcji wyświetliłam imiona i nazwiska wszystkich influencerów z tabeli Influencerzy w osobnej kolumnie *FullName*. Ostatnio stworzona funkcja natomiast policzyła wartość wysyłki w innej walucie niż pierwotnie, czyli zamieniła złotówki na euro z dokładnością do 2 miejsc po przecinku.

| SELECT sh.Id\_shipping, FullName(inf.first\_name, inf.last\_name) AS FullName, sh.Product\_Id\_product, sh.date, pr.price, ValueToEuro(pr.price, 4.53) AS EUR  FROM influencer AS inf, shipping AS sh, product AS pr  WHERE inf.Id\_influencer = sh.Influencer\_Id\_influencer  AND pr.Id\_product = sh.Product\_Id\_product; |
| --- |



## Procedury

Procedury ograniczają liczbę danych, które są przesyłane między serwerem bazy danych a klientem. Ograniczają więc również obciążenie serwera bazy danych, gdyż serwer taki realizuje mniej połączeń.

Zarówno funkcja jak i procedura to ręcznie wykonywane programy. Można je wyjaśnić na podstawie analogii: jeśli chciałabym odczytać pensję influencera, to skorzystałabym z funkcji. Jeśli jednak chciałabym ją zmodyfikować, to skorzystałabym z procedury. To tak jak *gettery* i *settery* w językach programowania.

Funkcje od procedur różnią się głównie tym, że funkcja musi zwracać wartość (w klauzuli return), a procedura może (za pomocą parametru out) ale nie musi. Dodatkowo, funkcje mogą być wywoływane z procedury ale procedura nie może być wywołana z funkcji. Procedury mogą wykorzystywać/wykonywać polecenia SQL, a funkcje niekoniecznie.

### Wyświetlanie danych o influencerach

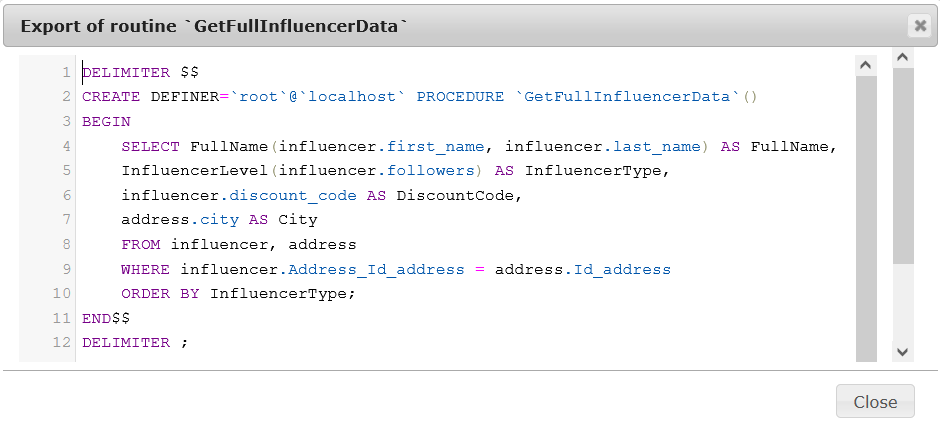
W oparciu o wcześniej utworzone funkcje, procedura ta wyświetla podstawowe dane o wszystkich influencerach z tabeli Influencerzy, imię i nazwisko (korzystając z funkcji FullName()), etykietę influencera (korzystając z funkcji InfluencerLevel()), jego kod zniżkowy oraz miasto, z którego pochodzi.

Procedura GetFullInfluencerData() nic nie pobiera ani nie zwraca. Wyświetla jedynie dane za pomocą poleceń SQL umieszczonych w jej wnętrzu.

**POLECENIE SQL:**

| DELIMITER //  CREATE PROCEDURE GetFullInfluencerData()  BEGIN  SELECT FullName(influencer.first\_name, influencer.last\_name) AS FullName,  InfluencerLevel(influencer.followers) AS InfluencerType,  influencer.discount\_code AS DiscountCode,  address.city AS City  FROM influencer, address  WHERE influencer.Address\_Id\_address = address.Id\_address  ORDER BY InfluencerType;  END//  DELIMITER ; |
| --- |

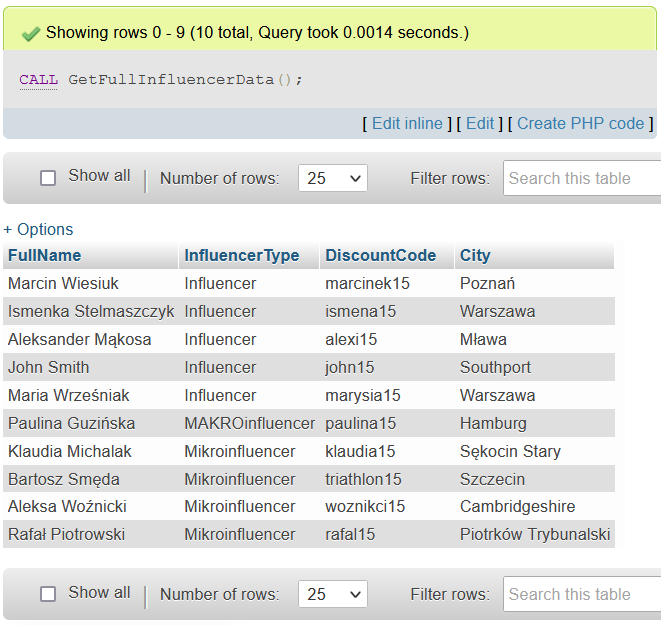
Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Utworzoną procedurę wywołałam poniższym poleceniem.

| CALL GetFullInfluencerData(); |
| --- |

Zrzut wyniku działania procedury GetFullInfluencerData() poniżej.



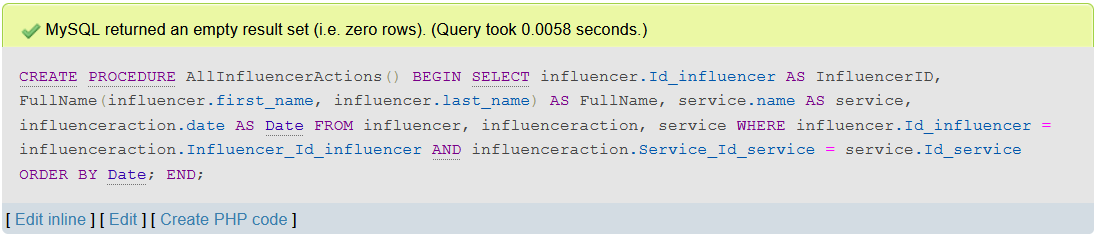
### Wyświetlanie działań wszystkich influencerów

Korzystając z wcześniej utworzonych funkcji, procedura ta wyświetla wszystkie usługi jakie wykonali wszyscy influencerzy i sortuje je rosnąco (domyślnie) według daty świadczenia usługi.

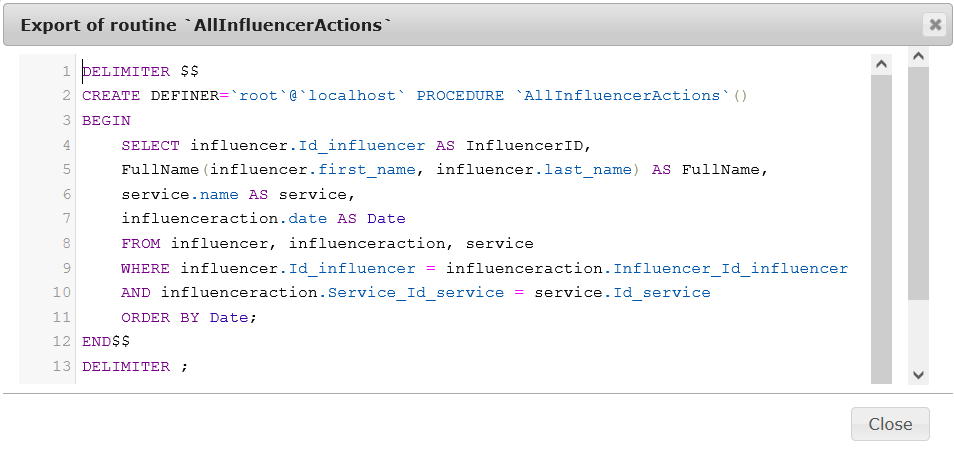
Procedura AllInfluencerActions() wyświetla dane za pomocą poleceń SQL umieszczonych w jej wnętrzu, nic nie pobiera ani nie zwraca.

**POLECENIE SQL:**

| DELIMITER //  CREATE PROCEDURE AllInfluencerActions()  BEGIN  SELECT influencer.Id\_influencer AS InfluencerID,  FullName(influencer.first\_name, influencer.last\_name) AS FullName,  service.name AS service,  influenceraction.date AS Date  FROM influencer, influenceraction, service  WHERE influencer.Id\_influencer = influenceraction.Influencer\_Id\_influencer  AND influenceraction.Service\_Id\_service = service.Id\_service  ORDER BY Date;  END//  DELIMITER ; |
| --- |



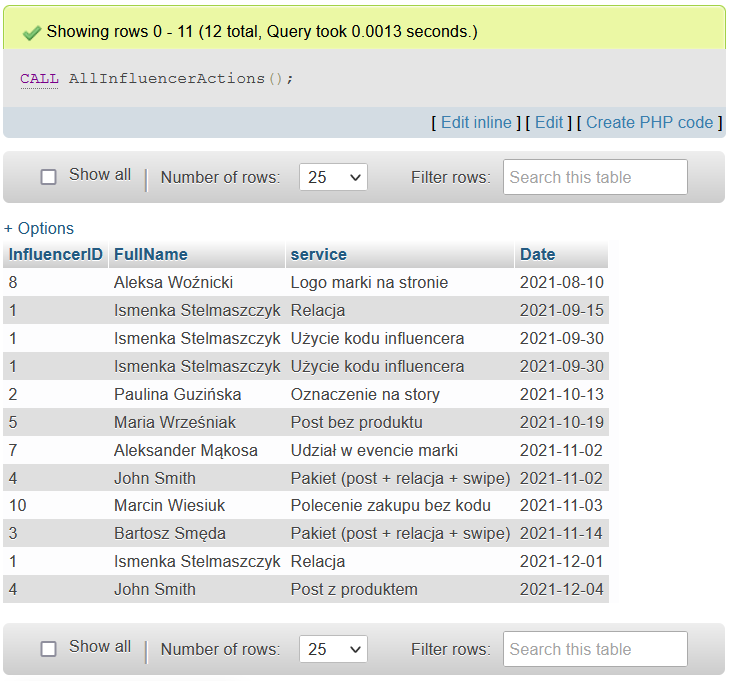
Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Utworzoną procedurę wywołałam poniższym poleceniem.

| CALL AllInfluencerActions(); |
| --- |

Zrzut wyniku działania procedury GetFullInfluencerData().

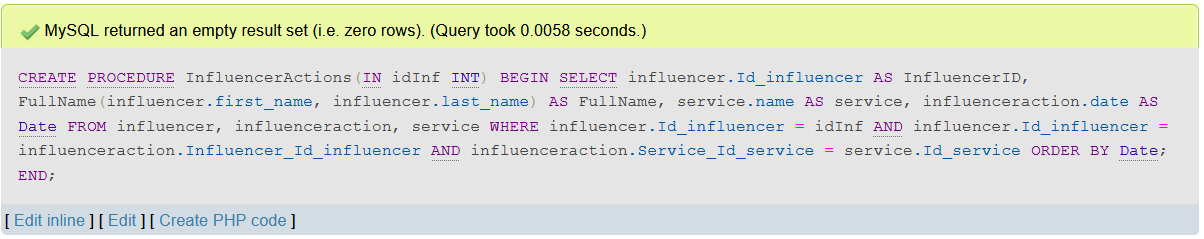


### Wyświetlanie działań konkretnego influencera

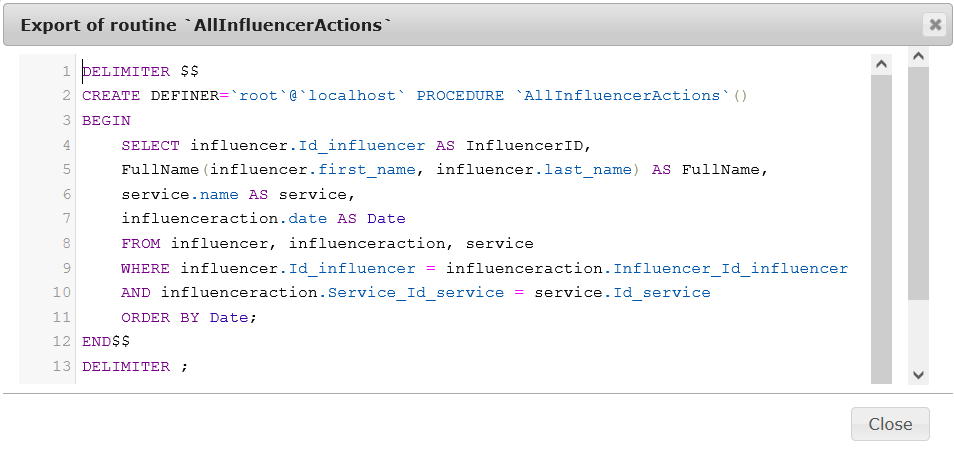
Udoskonalenie poprzedniej procedury o wyświetlanie świadczonych usług zawężonych do jednego konkretnego influencera. Procedura InfluencerActions() pobiera tym razem ID influencera.

**POLECENIE SQL:**

| DELIMITER //  CREATE PROCEDURE InfluencerActions(IN idInf INT)  BEGIN  SELECT influencer.Id\_influencer AS InfluencerID,  FullName(influencer.first\_name, influencer.last\_name) AS FullName,  service.name AS service,  influenceraction.date AS Date  FROM influencer, influenceraction, service  WHERE influencer.Id\_influencer = idInf  AND influencer.Id\_influencer = influenceraction.Influencer\_Id\_influencer  AND influenceraction.Service\_Id\_service = service.Id\_service  ORDER BY Date;  END//  DELIMITER ; |
| --- |

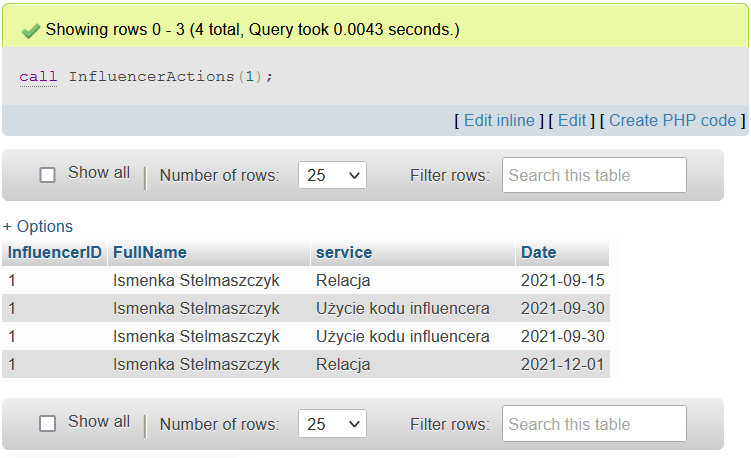


Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Wywołałam procedurę z argumentem 1 do wyświetlenia usług influencera o ID równym 1.

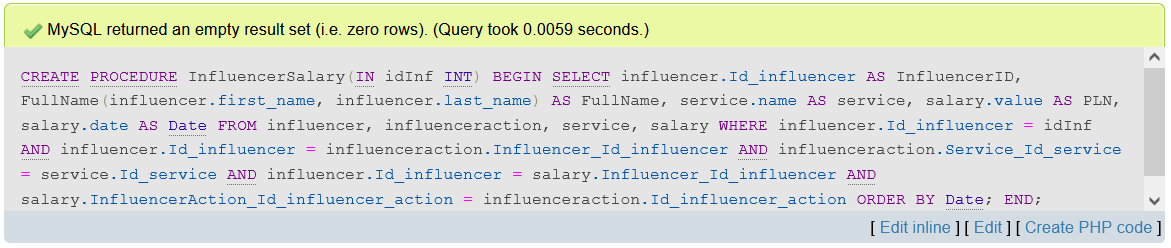
| CALL InfluencerActions(1); |
| --- |



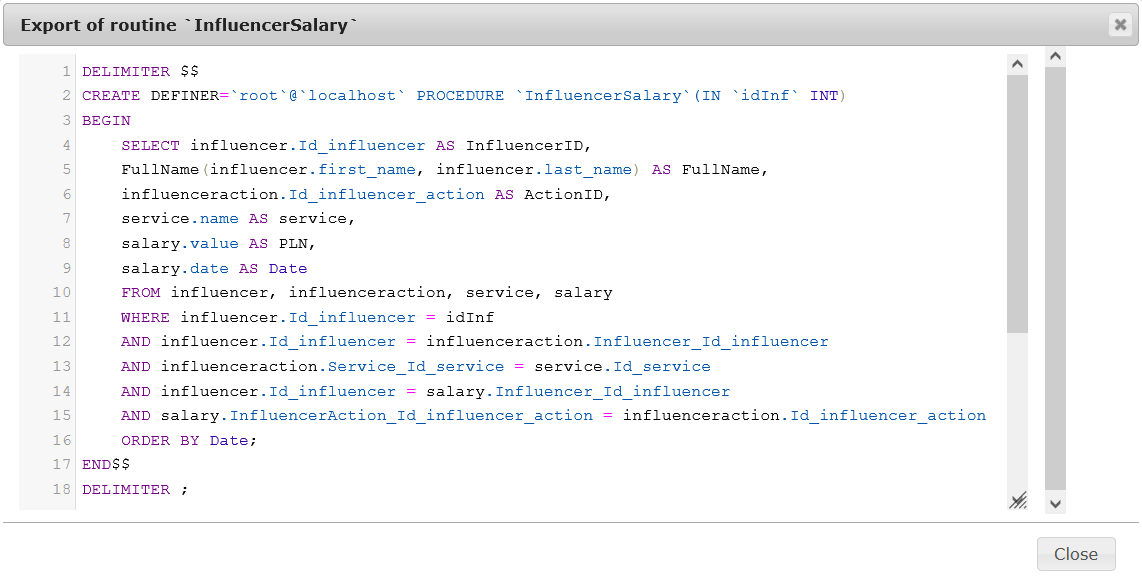
### Wyświetlanie wynagrodzenia dla influencera w danym miesiącu

**POLECENIE SQL:**

| DELIMITER //  CREATE PROCEDURE InfluencerSalary(IN idInf INT)  BEGIN  SELECT influencer.Id\_influencer AS InfluencerID,  FullName(influencer.first\_name, influencer.last\_name) AS FullName,  service.name AS service,  salary.value AS PLN,  salary.date AS Date  FROM influencer, influenceraction, service, salary  WHERE influencer.Id\_influencer = idInf  AND influencer.Id\_influencer = influenceraction.Influencer\_Id\_influencer  AND influenceraction.Service\_Id\_service = service.Id\_service  AND influencer.Id\_influencer = salary.Influencer\_Id\_influencer  AND salary.InfluencerAction\_Id\_influencer\_action = influenceraction.Id\_influencer\_action  ORDER BY Date;  END//  DELIMITER ; |
| --- |



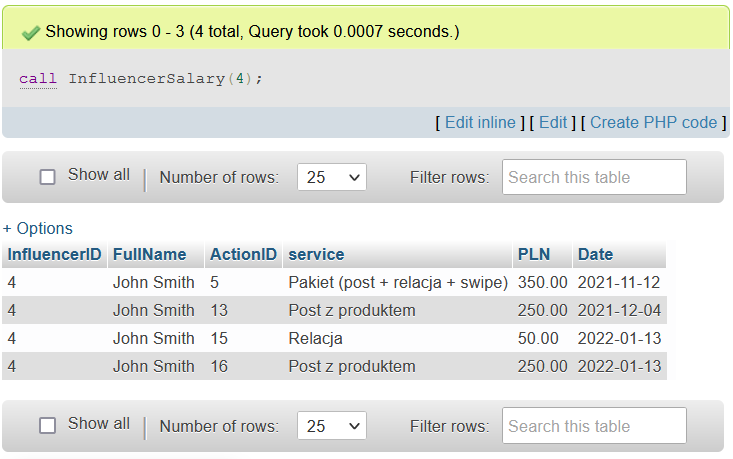
Poniżej dodatkowo podgląd wyeksportowanej funkcji.



Wywołałam procedurę z argumentem 1 do wyświetlenia usług influencera o ID równym 1.

| call InfluencerSalary(4); |
| --- |

Zrzut wyniku działania procedury dla zarobków influencera o ID równym 4.



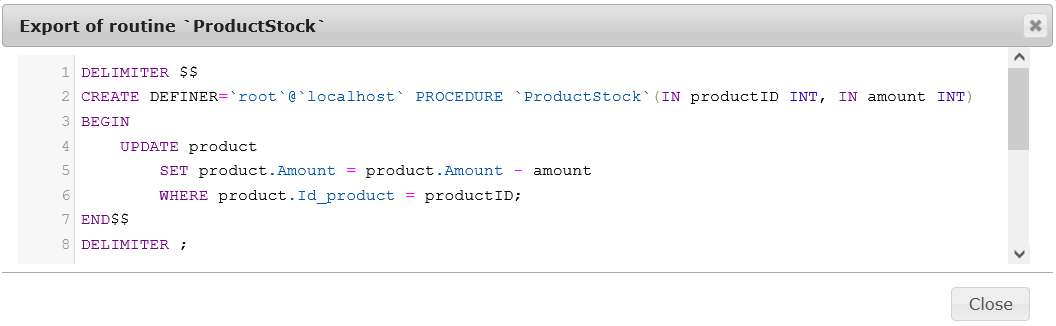
### Zmniejszenie ilość towarów w magazynie

Podaję jako parametry wejściowe ID produktu oraz jego ilość. Procedura aktualizuje ilość produktu o danym ID zmniejszając jego ilość w magazynie (tabela *product*) o taką, jaką podamy w drugim parametrze.

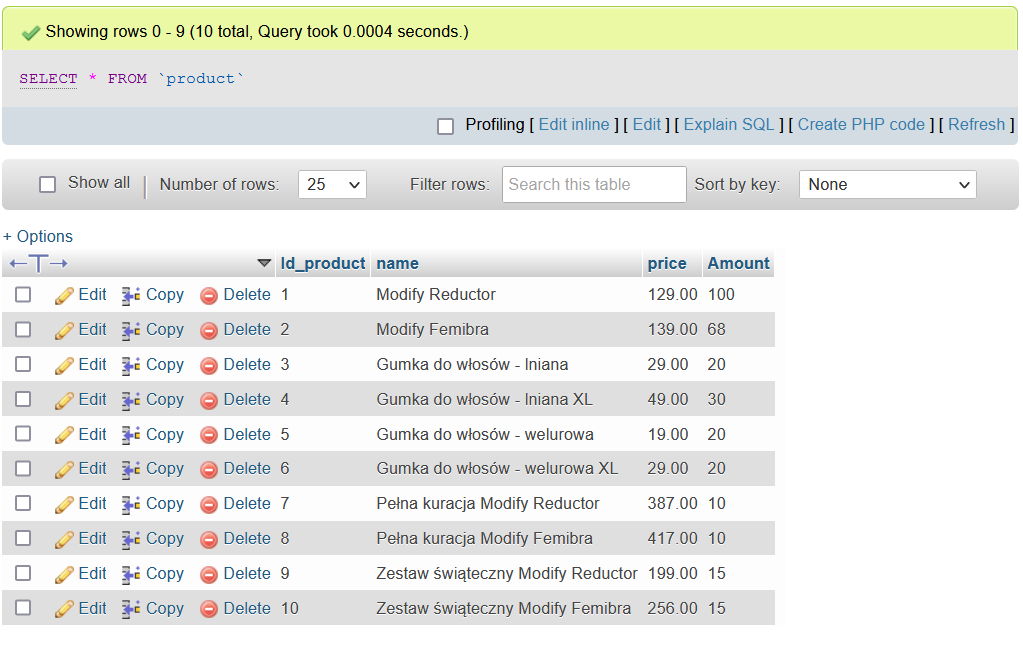
**POLECENIE SQL:**

| DELIMITER //  CREATE PROCEDURE ProductStock(IN productID INT, IN amount INT)  BEGIN  UPDATE product  SET product.Amount = product.Amount - amount  WHERE product.Id\_product = productID;  END//  DELIMITER ; |
| --- |

Poniżej podgląd wyeksportowanej funkcji.

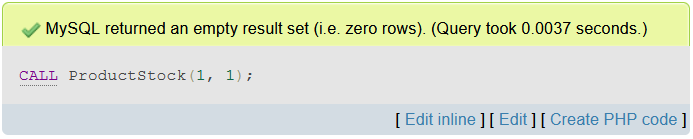


Dla przetestowania działania procedury wywołałam ją na jednym z produktów. Najpierw poniżej screen przed wywołaniem.

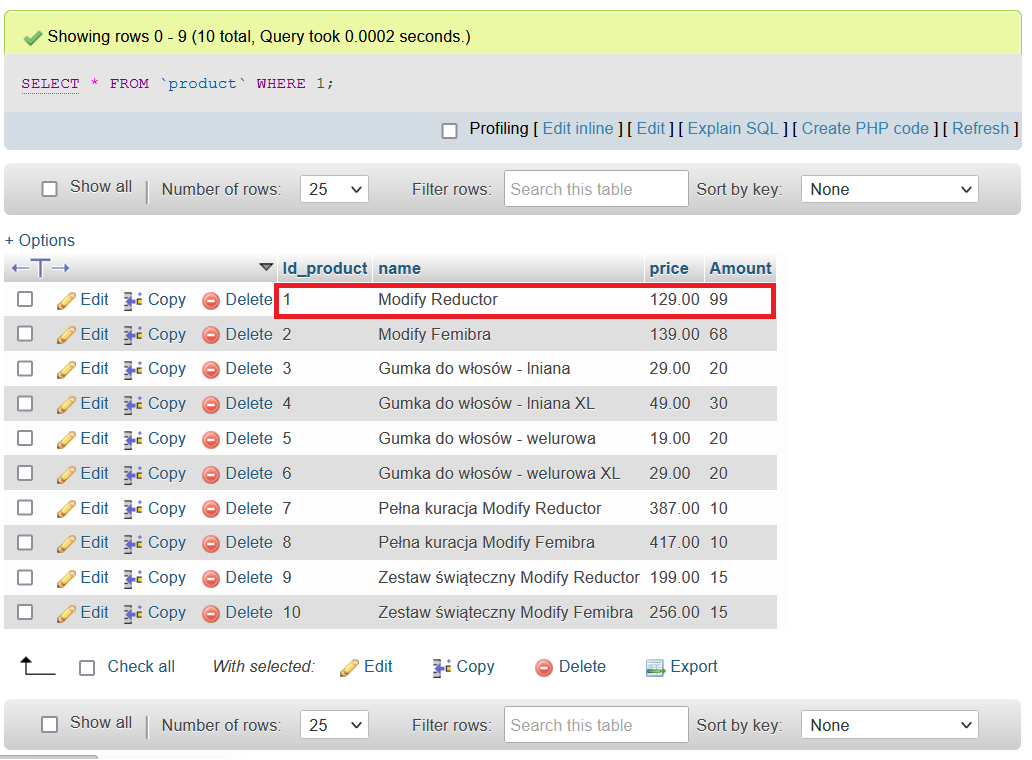


Wywołałam procedurę z argumentami: 1 jako ID produktu (Modify Reductor) oraz 1 jako ilością tego produktu do zmniejszenia.

| CALL ProductStock(1, 1); |
| --- |



Zrzut po wywołaniu procedury zmniejszającej ilość produktu o 1. Zgodnie z oczekiwaniami ilość danego produktu zmniejszyła się o 1: ze 100 na 99.



## Obsługa wyjątków

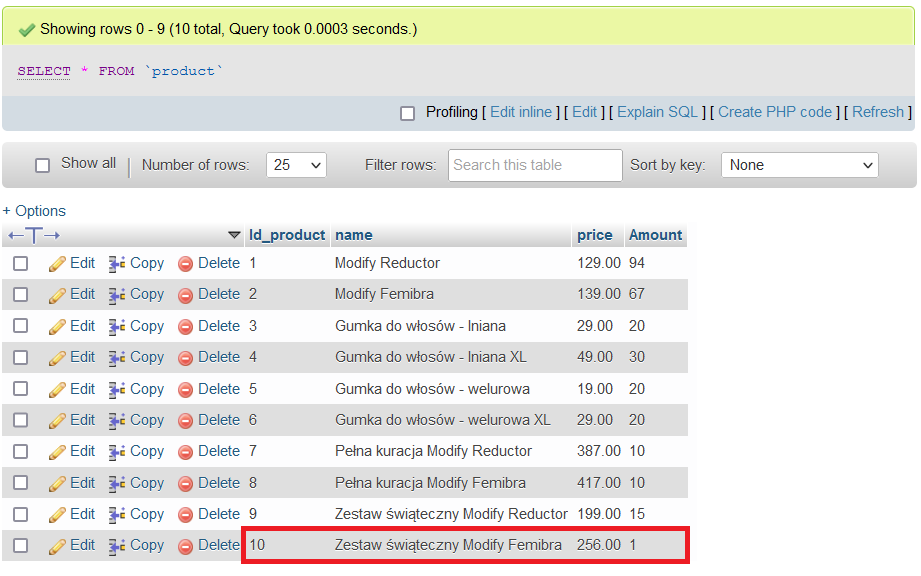
Powyższą procedurę można poprawić dodając do niej obsługę wyjątku. Obecnie, w momencie, kiedy stan magazynu jest równy 0, procedura i tak zmniejszy ilość produktu do wartości ujemnej. Aby temu zapobiec, dodam do kodu fragment przechwytujący taką sytuację - informujący o błędzie i nie zmniejszający ilości produktu.

**POLECENIE SQL:**

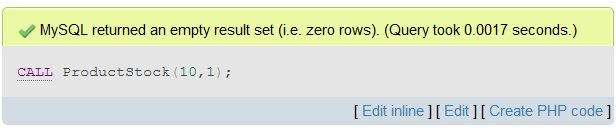
| DELIMITER $$  CREATE DEFINER=`root`@`localhost` PROCEDURE `ProductStock`(IN `productID` INT, IN `amount` INT)  BEGIN  SET @MESSAGE\_TEXT = 'The state of a stock is 0 for this product!';  SET @GET\_PRODUCT\_AMOUNT = (SELECT product.Amount FROM product WHERE product.Id\_product = productID LIMIT 1);  IF @GET\_PRODUCT\_AMOUNT - amount <= 0 THEN  SELECT @MESSAGE\_TEXT;  ELSE  UPDATE product  SET product.Amount = product.Amount - amount  WHERE product.Id\_product = productID;  END IF;  END$$  DELIMITER ; |
| --- |

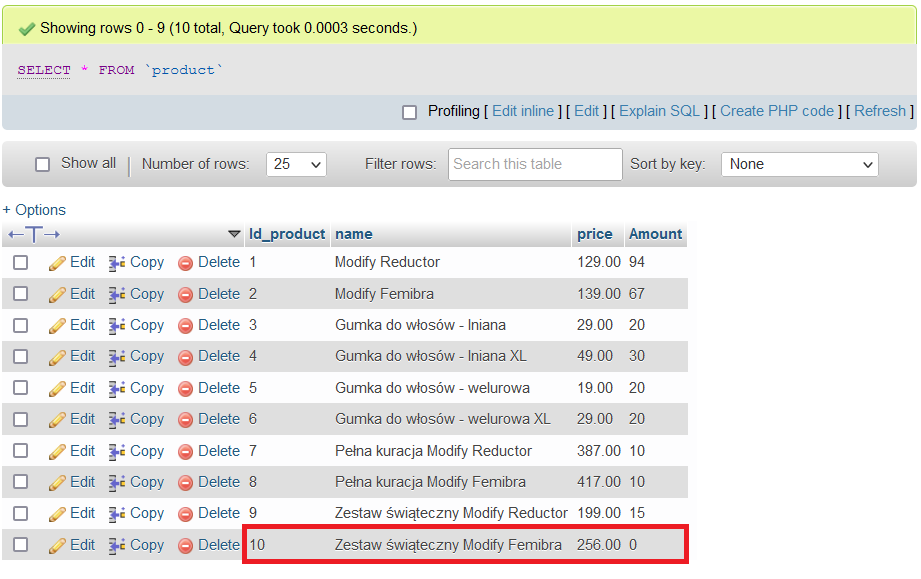
## 

Zmieniłam ilość produktu o ID równym 10 na 1 aby pokazać działanie procedury.

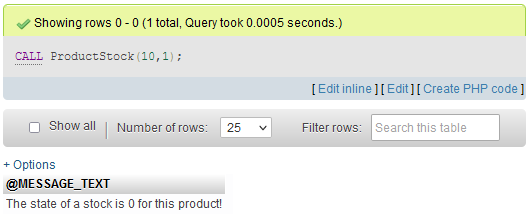


Za pierwszym razem, kiedy wywołuję procedurę na produkcie o ID=10 oraz z ilością 1, ze stanu magazynu zostaje zdjęta oczekiwana ilość produktu i zostaje na stanie 0 sztuk. Screen poniżej.





Kiedy na tym samym produkcie wywołam tę samą procedurę ponownie, tym razem stan magazynu jest równy zero dla produktu o ID równym 10 a więc otrzymam odpowiedni komunikat z procedury. Jak widać w kodzie procedury, jeśli spróbowałabym zdjąć ze stanu dowolną ilość większą niż jest, również otrzymam komunikat błędu.



Stan magazynu dla tego produktu nie uległ zmianie. Obsługa wyjątków działa prawidłowo.

## Wyzwalacze

Zdarzenie, inaczej trigger, wyzwalacz, to skrypt (fragment kodu) wykonywany automatycznie w przypadku zajścia jakiegoś zdarzenia w bazie danych (np. dodania danych, ich modyfikacji, czy usunięcia).

Istnieje kilka typów wyzwalaczy. najpowszechniejsze to BEFORE i AFTER. Dla każdego typu istnieją trzy zdarzenia powodujące wykonanie wyzwalacza i są to:

* 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 operacją usunięcia rekordu
* BEFORE INSERT – wykonanie wyzwalacza przed dodaniem rekordu
* BEFORE UPDATE – wykonanie wyzwalacza przed zmodyfikowaniem rekordu

### Wywołanie procedury zmniejszania ilości magazynu

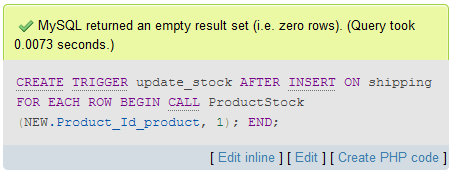
Za każdym razem, kiedy stworzona zostanie nowa wysyłka (rekord w tabeli *shipping)*, wyzwalacz wywoła procedurę zmniejszającą w magazynie ilość wysyłanego produktu.

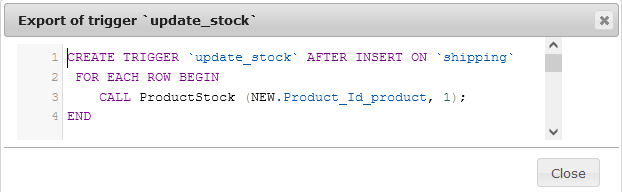
Skorzystałam w wcześniej stworzonej procedury ProductStock, która przyjmuje w parametrach ID produktu oraz jego ilość. Procedura ta aktualizuje ilość produktu o danym ID zmniejszając jego ilość w magazynie (tabela *product*) o taką, jaką podamy w drugim parametrze.

Następnie utworzyłam wyzwalacz do uruchomienia tej procedury po każdorazowym dodaniu rekordu do tabeli *shipping*.

**POLECENIE SQL:**

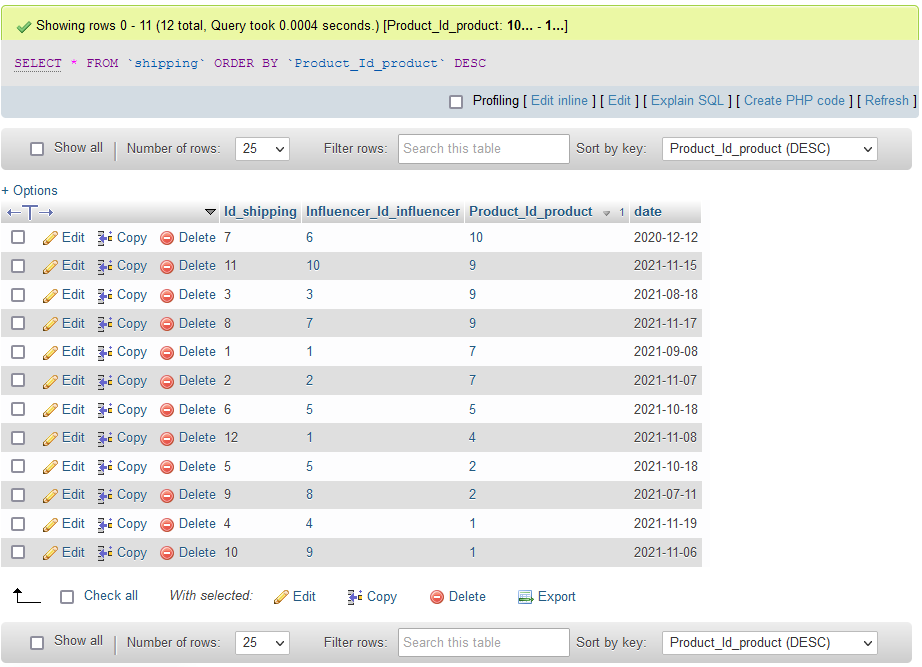
| DELIMITER //  CREATE TRIGGER update\_stock  AFTER INSERT ON shipping  FOR EACH ROW  BEGIN  CALL ProductStock (NEW.Product\_Id\_product, 1);  END//  DELIMITER ; |
| --- |

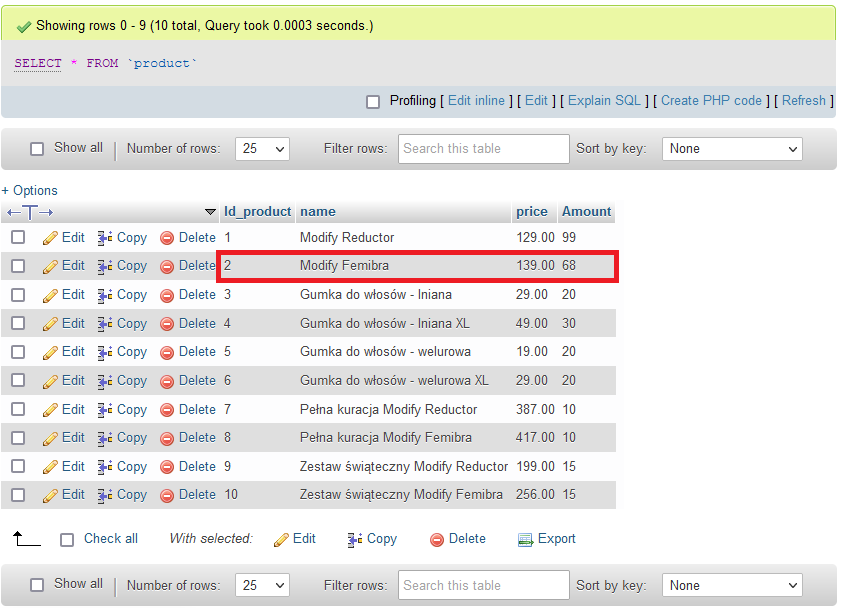




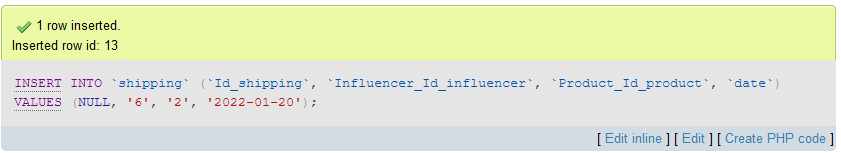
Po pomyślnym utworzeniu triggera wywołującego procedurę, sprawdziłam działanie. Utworzyłam wysyłkę do influencera z konkretnym produktem, czyli dodaję rekord w tabeli *shipping*. Po dodaniu rekordu powinien wywołać się trigger uruchamiający procedurę zmniejszania ilości magazynu (tabeli *product*) o wysłany produkt. Screeny z tego działania w krokach poniżej.

Widok tabeli *shipping* i *product* PRZED operacją.

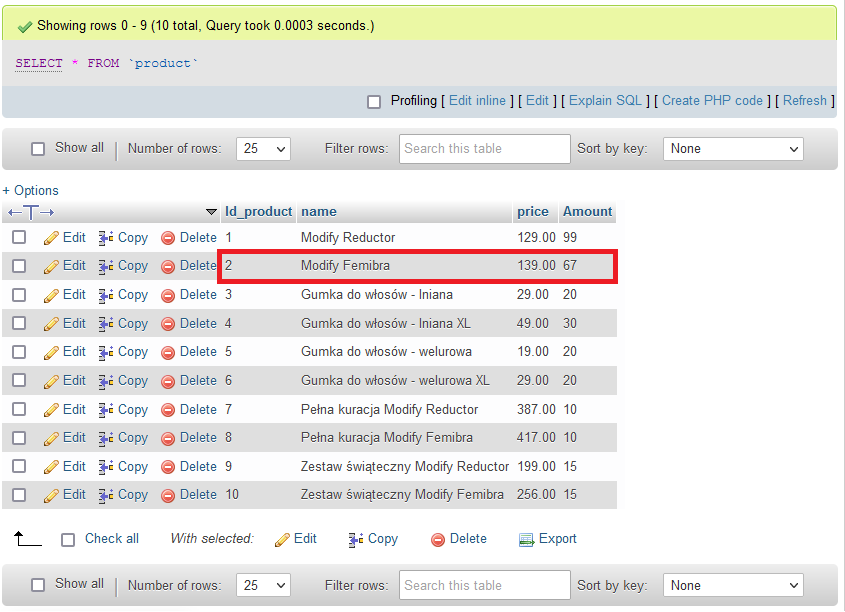
****

****

Dodałam wysyłkę, a więc rekord do tabeli *shipping*.



Widok tabeli *product* PO operacji. Ilość wysyłanego produktu zmniejszyła się w magazynie zgodnie z oczekiwaniami.



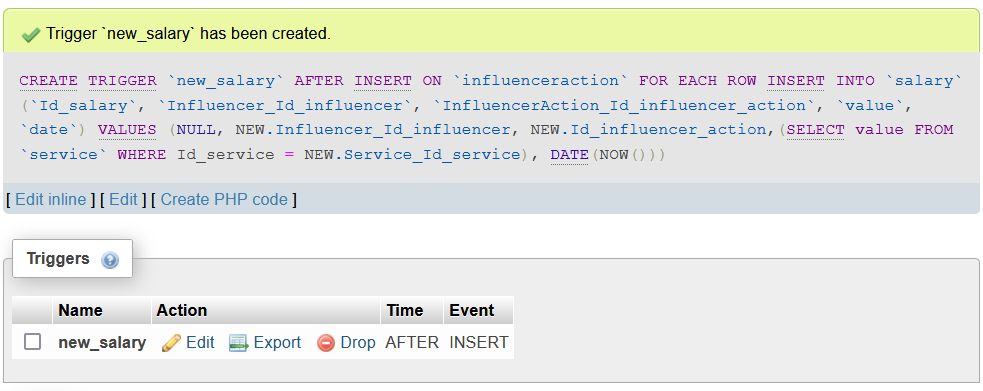
### Dodanie nowego rekordu

Dodaję trigger tworzenia nowego rekordu w tabeli *salary*. Będzie się on wywoływać za każdym razem, gdy dodam akcję do influencera, czyli nowy rekord w tabeli *influenceraction*.

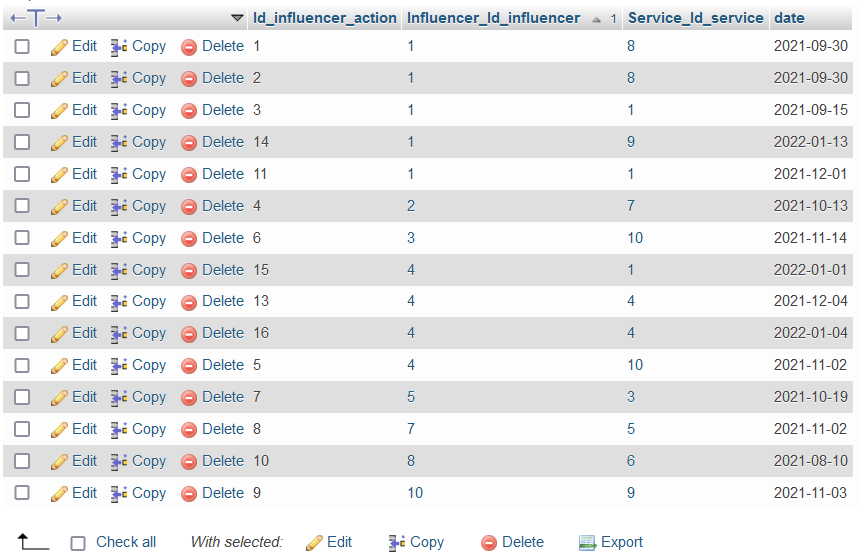
Automatycznie do tabeli salary

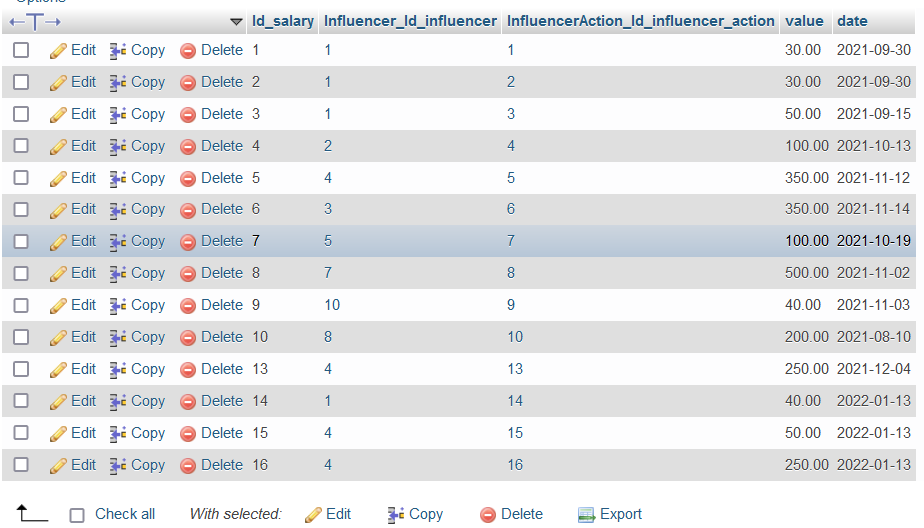
**POLECENIE SQL:**

| CREATE TRIGGER `new\_salary`  AFTER INSERT ON `influenceraction`  FOR EACH ROW  INSERT INTO `salary` (`Id\_salary`, `Influencer\_Id\_influencer`, `InfluencerAction\_Id\_influencer\_action`, `value`, `date`)  VALUES (NULL, NEW.Influencer\_Id\_influencer, NEW.Id\_influencer\_action, (SELECT value FROM `service` WHERE Id\_service = NEW.Service\_Id\_service), DATE(NOW())) |
| --- |

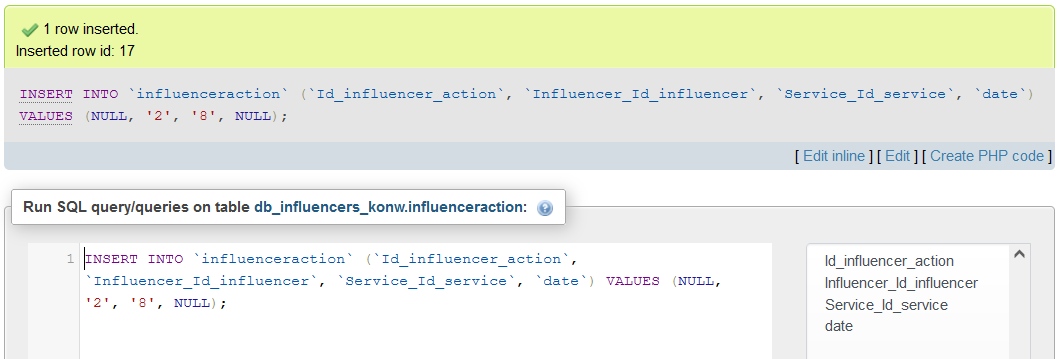


Dla sprawdzenia działania dodałam rekord w tabeli “influenceraction”. Screen tabel *influenceraction* i *salary* przed operacją:

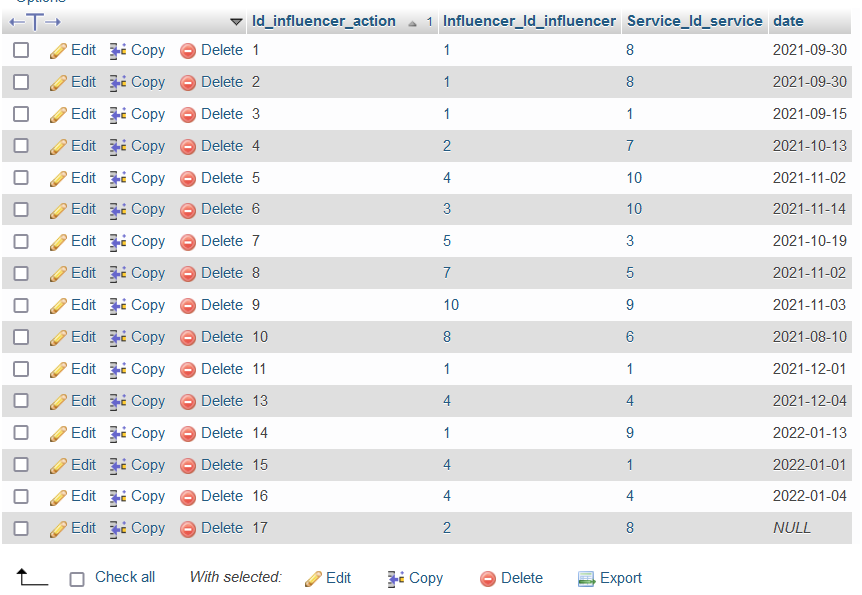




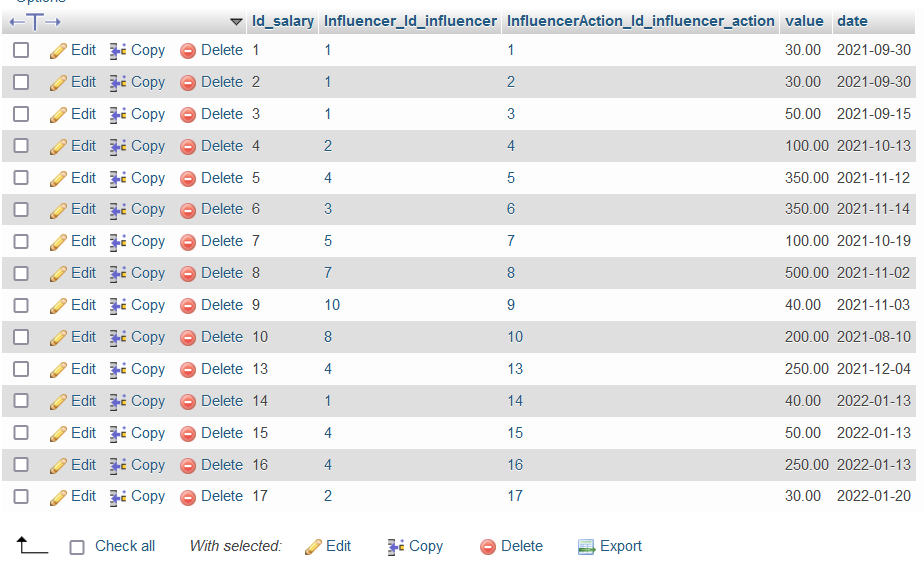
oraz po dodaniu rekordu w tabeli *influenceraction*:



widać, że w tabeli *influenceraction* dodano 1 rekord (o ID=17)



a tym samy automatycznie w tabeli *salary* stworzony został rekord o wynagrodzeniu dla influencera za dodaną wyżej akcję



Wyzwalacz wygenerował w tabeli “Salary” rekord z automatycznie zaciągniętą wartością usługi z tabeli “service” co świadczy o poprawności działania skryptu zdarzenia.

## Przydzielanie uprawnień użytkownikom

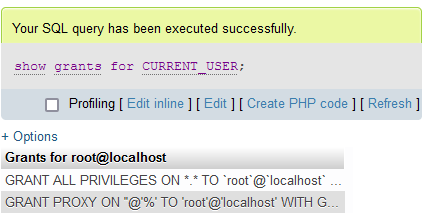
System MySQL jak każdy inny system do zarządzania bazami danych pomaga użytkownikom przechowywać, uporządkowywać i pobierać dane. Ze względów bezpieczeństwa zalecane jest tworzenie różnego typu użytkowników MySQL o odrębnych uprawnieniach w celu ograniczenia dostępu i uniknięcia niepożądanych zmian w bazach danych.

**Sprawdzenie bieżących uprawnień**

Przed tworzeniem nowych użytkowników i nadawaniu im uprawnień upewniłam się, że sama mam do tego uprawnienia będąc na obecnym koncie.

**POLECENIE SQL:**

| SHOW GRANTS FOR CURRENT\_USER; |
| --- |

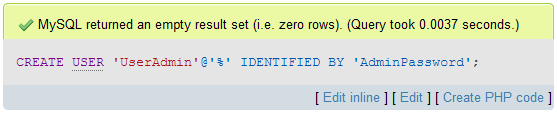


### Tworzenie nowego użytkownika z uprawnieniami READONLY

Utworzyłam nowego użytkownika przez polecenia w MySQL:

**POLECENIE SQL:**

| CREATE USER 'UserAdmin'@'%' IDENTIFIED BY 'AdminPassword'; |
| --- |



**Nadawanie uprawnień**

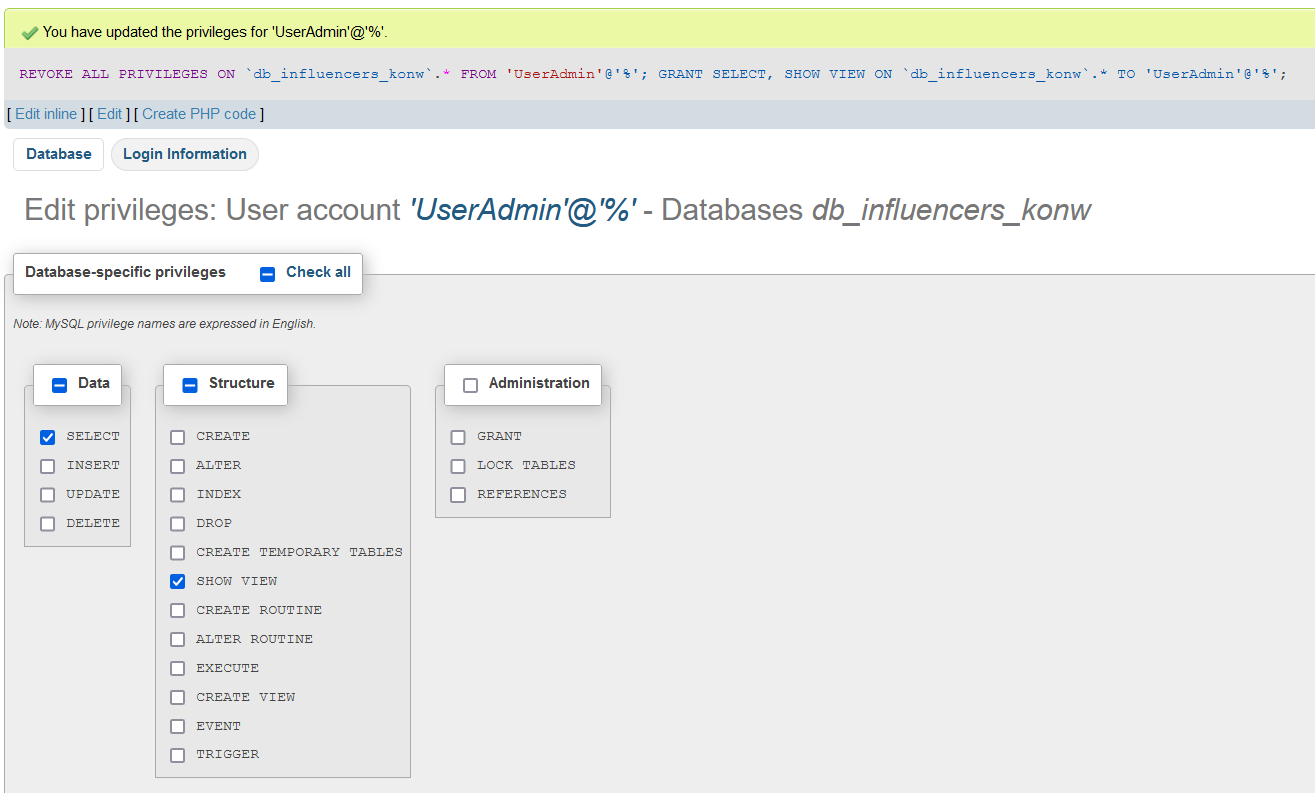
W tym miejscu nowy użytkownik o nazwie “UserAdmin” nie jest jeszcze upoważniony do wniesienia jakichkolwiek zmian w bazę danych. W rzeczywistości, jeśli nowy użytkownik spróbowałby się na tym etapie zalogować (z hasłem 'AdminPassword'), nie będzie on mógł dostać się do linii poleceń MySQL. Dlatego, najpierw muszę mu zapewnić dostęp do informacji, do której będzie mógł się dostać. Nadaję mu 2 uprawnienia typu readonly.

**POLECENIE SQL:**

| GRANT SELECT, SHOW VIEW ON ‘db\_influencers\_konw’.\* TO ‘UserAdmin’@’%’; |
| --- |

* Uprawnienia SELECT, SHOW VIEW umożliwią użytkownikom MySQL dostęp do tych właśnie poleceń (i żadnych innych),
* W powyższym wyrażeniu ‘%’ oznacza, że użytkownik będzie miał dostęp ze wszystkich adresów IP,
* \* oznacza wszystkie.

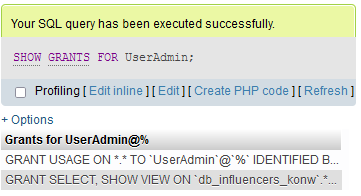
Poniżej widoczne dodakowo nadane uprawnienia tylko do odczytu.



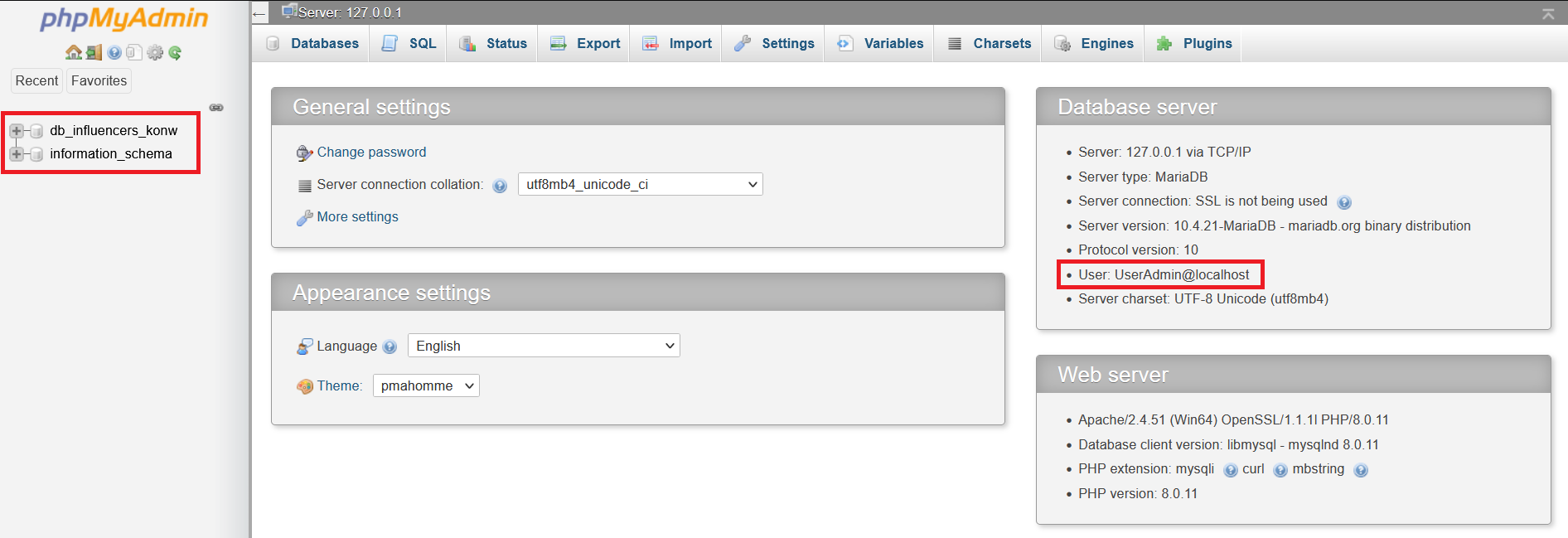
Upewniam się, jakie przywileje zostały nadane.

**POLECENIE SQL:**

| SHOW GRANTS FOR UserAdmin; |
| --- |



Loguję się jako nowo utworzony użytkownik. Widać, że jedyna baza, do której jest dostęp to ta, do której udzieliłam dostępu czyli “db\_influencers\_konw“.



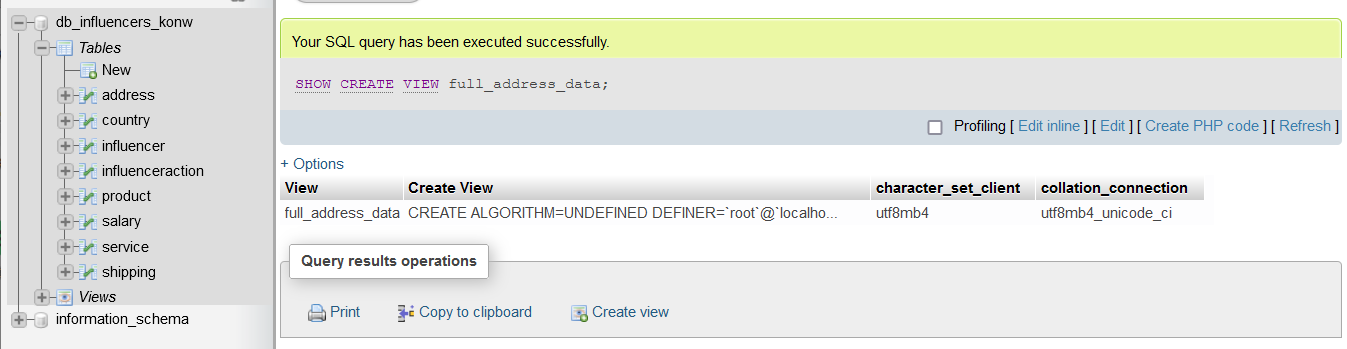
Uprawnień udzieliłam jedynie dla polecenia SELECT oraz SHOW VIEW. Jak widać na poniższym screenie, polecenie SELECT działa bez zarzutu.



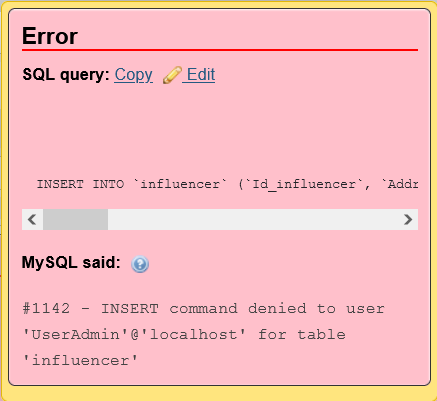
Bez zarzutu działa również “czytanie” definicji utworzonych widoków. To druga z opcji typu readonly, która działa według oczekiwań, ponieważ nadany został taki przywilej dla tego konta.

**POLECENIE SQL:**

| SHOW CREATE VIEW full\_address\_data; |
| --- |



Tym razem spróbuję dodać jakiś rekord do dowolnej tabeli. Polecenie INSERT nie było nadane dla tego konta w przywilejach. Jak widać poniżej, jest odmowa wykonania go dla tego użytkownika. Wszystko działa zgodnie z założeniami.

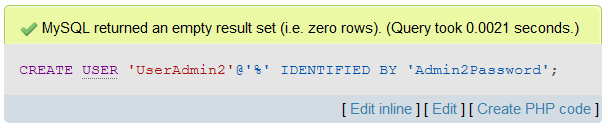


### Tworzenie nowego użytkownika z uprawnieniami ‘full access’

Utworzyłam nowego użytkownika przez polecenia w MySQL:

**POLECENIE SQL:**

| CREATE USER 'UserAdmin2'@'%' IDENTIFIED BY 'Admin2Password'; |
| --- |



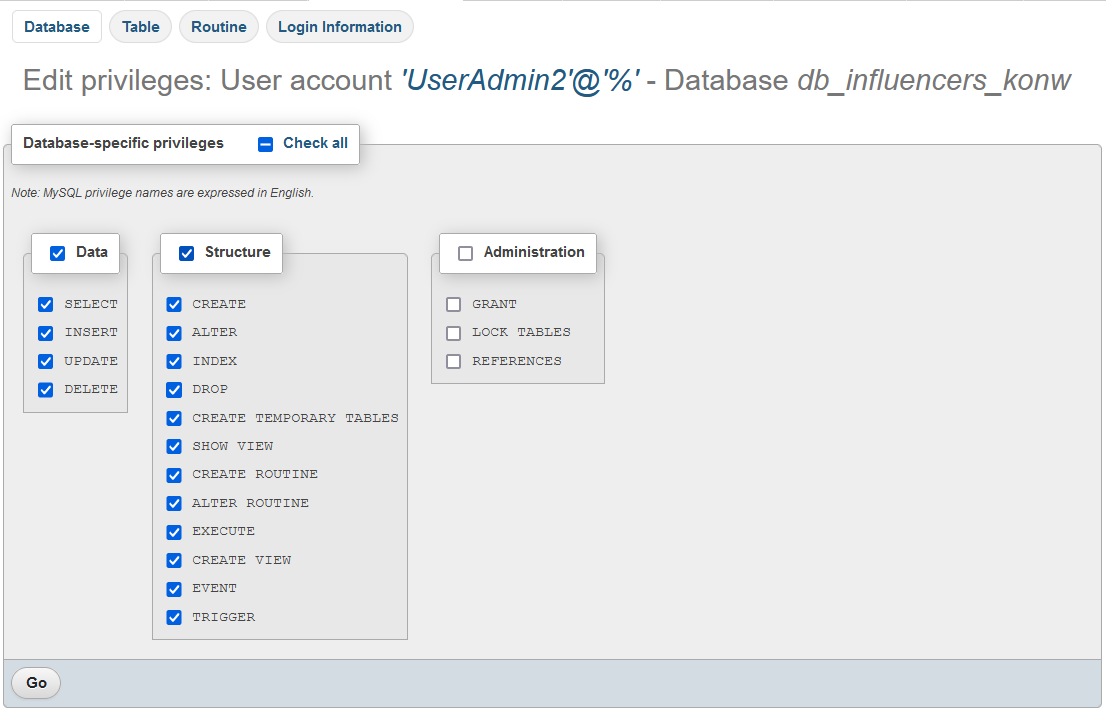
**Nadawanie uprawnień**

Tak jak poprzednio, zapewniam użytkownikowi najpierw dostęp do informacji, do której będzie mógł się dostać. Tym razem nadaję mu pełny dostęp do bazy danych ‘db\_influencers\_konw’ - poza opcjami administrującymi.

**POLECENIE SQL:**

| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE TEMPORARY TABLES, CREATE VIEW, EVENT, TRIGGER, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, EXECUTE ON `db\_influencers\_konw`.\* TO 'UserAdmin2'@'%'; |
| --- |

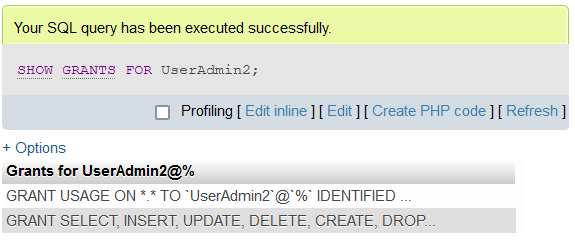
Poniżej widoczne dodatkowo nadane wszystkie uprawnienia - poza administracyjnymi.



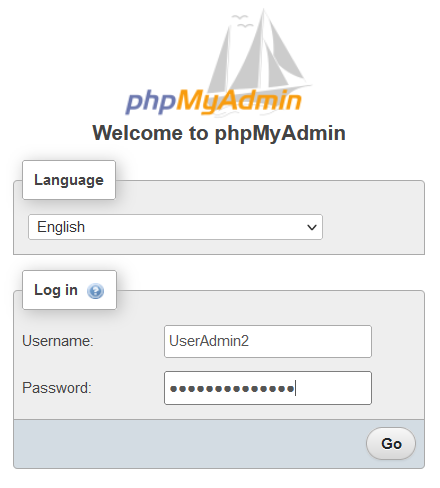
Upewniam się, jakie przywileje zostały nadane.

**POLECENIE SQL:**

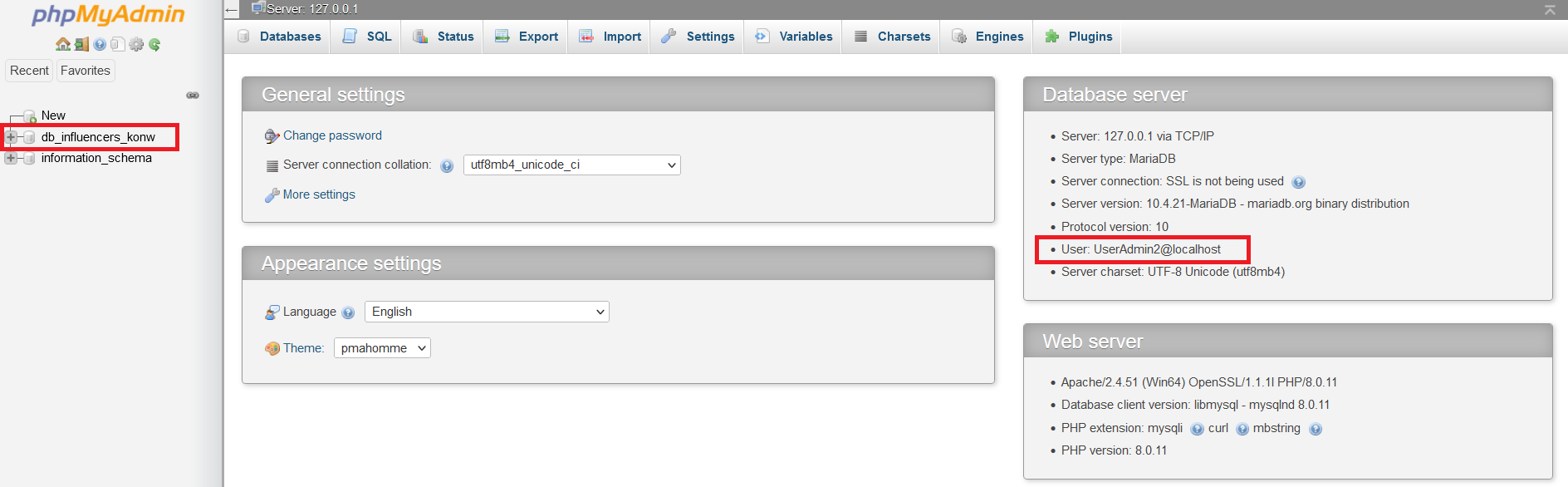
| SHOW GRANTS FOR UserAdmin; |
| --- |



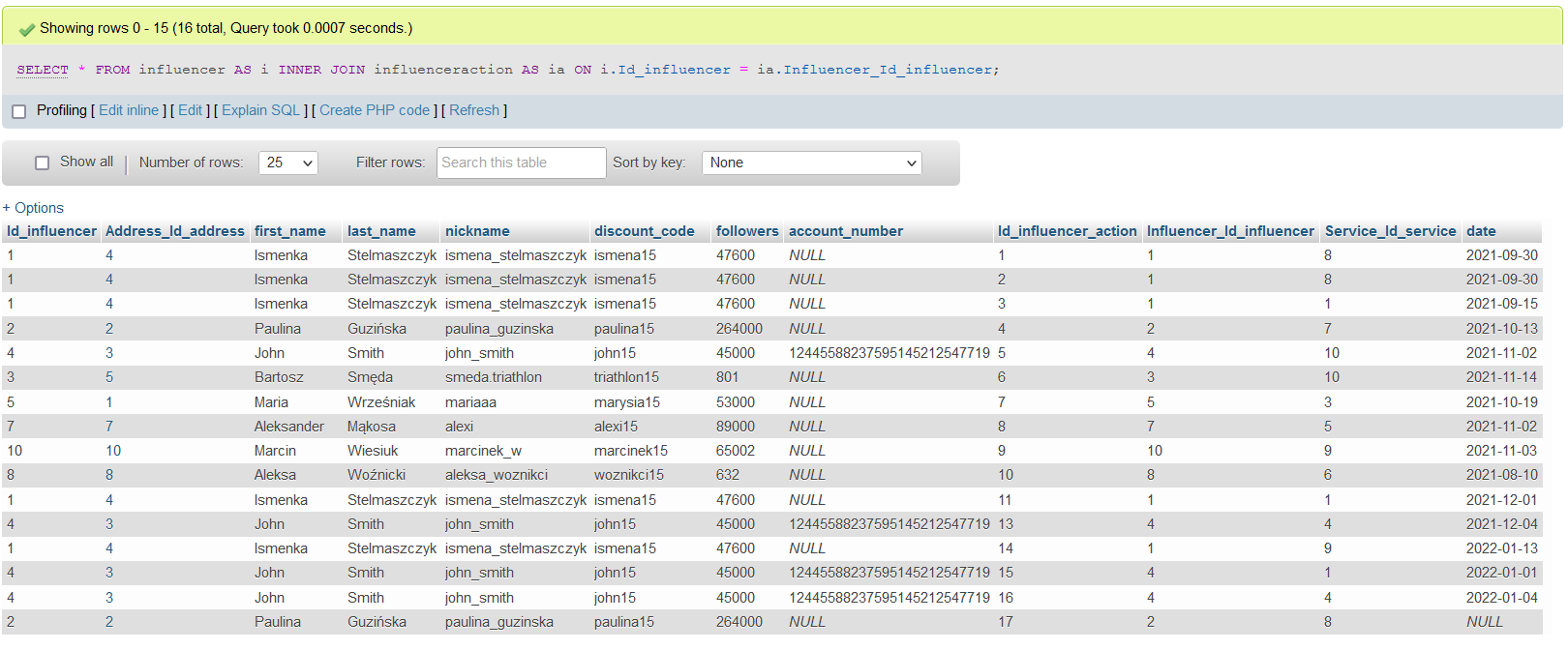
Loguję się jako nowo utworzony użytkownik UserAdmin2.



Widać, że jedyna baza, do której jest dostęp to ta, do której udzieliłam dostępu czyli “db\_influencers\_konw“.



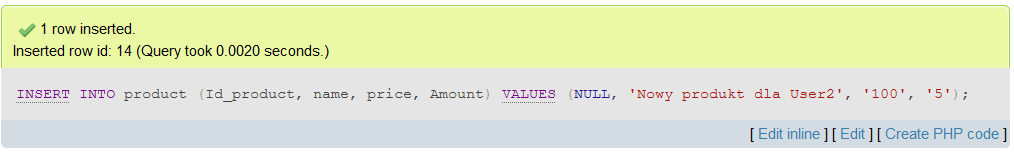
Uprawnień udzieliłam na wszystkie polecenia zapisu, odczytu i wszelkich modyfikacji danych (poza administracyjnymi). Jak widać na poniższym screenie, polecenie SELECT działa bez zarzutu.

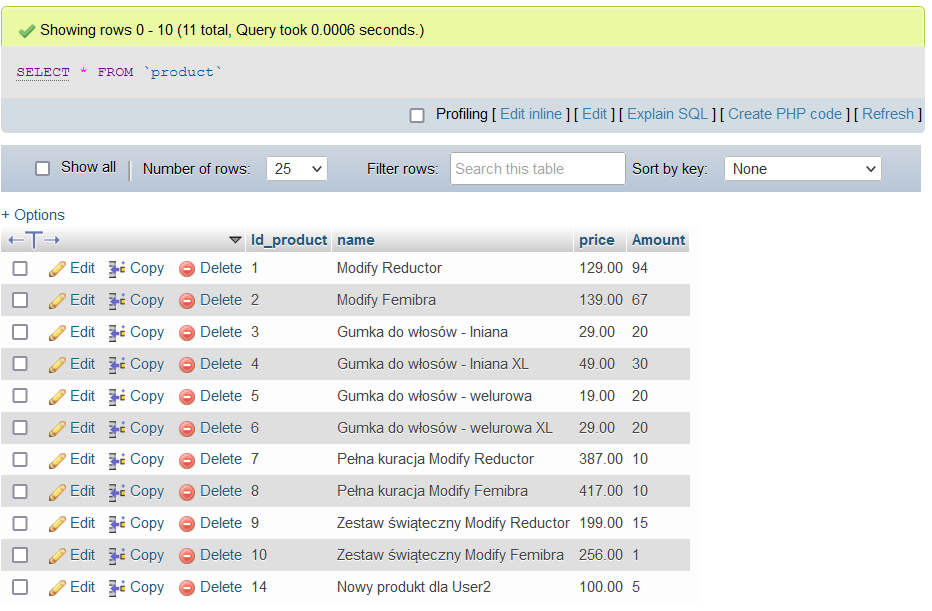


Bez zarzutu działa również polecenie INSERT, aby dodać jakiś rekord do dowolnej tabeli.

**POLECENIE SQL:**

| INSERT INTO product (Id\_product, name, price, Amount) VALUES (NULL, 'Nowy produkt dla User2', '100', '5'); |
| --- |





Jak widać, stworzony użytkownik posiada przywileje, które zostały mu nadane.

# Pełny skrypt SQL

| -- phpMyAdmin SQL Dump  -- version 5.1.1  -- https://www.phpmyadmin.net/  --  -- Host: 127.0.0.1  -- Generation Time: Jan 22, 2022 at 03:49 PM  -- Server version: 10.4.21-MariaDB  -- PHP Version: 8.0.11  SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";  START TRANSACTION;  SET time\_zone = "+00:00";  /\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  /\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  /\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  /\*!40101 SET NAMES utf8mb4 \*/;  --  -- Database: `db\_influencers\_konw`  --  DELIMITER $$  --  -- Procedures  --  $$  $$  $$  $$  $$  --  -- Functions  --  $$  $$  $$  $$  DELIMITER ;  -- --------------------------------------------------------  --  -- Table structure for table `address`  --  CREATE TABLE `address` (  `Id\_address` int(11) NOT NULL,  `Country\_Id\_country` int(11) DEFAULT NULL,  `street` varchar(200) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `house\_nr` varchar(10) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `post\_code` varchar(10) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `city` varchar(100) COLLATE utf8\_unicode\_ci DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `address`  --  INSERT INTO `address` (`Id\_address`, `Country\_Id\_country`, `street`, `house\_nr`, `post\_code`, `city`) VALUES  (1, 1, 'Puławska', '38/15', '03-232', 'Warszawa'),  (2, 2, 'Isern-Hinnerk-Weg', '14B', '22457', 'Hamburg'),  (3, 8, 'High Park Road', '20A', 'PR9 7QL', 'Southport'),  (4, 1, 'Jurajska', '4/59', '02-699', 'Warszawa'),  (5, 1, 'Kadłubka', '42/5', '71-524', 'Szczecin'),  (6, 1, 'Aleja Krakowska', '49', '05-090', 'Sękocin Stary'),  (7, 1, 'Nowowiejska', '15', '06-500', 'Mława'),  (8, 8, 'Saxonbury Way', '94', 'PE2 9FB', 'Cambridgeshire'),  (9, 1, 'Marii Dąbrowskiej', '96/9', '97-300', 'Piotrków Trybunalski'),  (10, 1, 'Niemeńska', '66', '60-412', 'Poznań'),  (11, NULL, 'Oakstreet', '23', '1', '890FR');  -- --------------------------------------------------------  --  -- Table structure for table `country`  --  CREATE TABLE `country` (  `Id\_country` int(11) NOT NULL,  `long\_name` varchar(200) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `short\_name` char(3) COLLATE utf8\_unicode\_ci DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `country`  --  INSERT INTO `country` (`Id\_country`, `long\_name`, `short\_name`) VALUES  (1, 'Polska', 'PL'),  (2, 'Niemcy', 'DE'),  (3, 'Rosja', 'RU'),  (4, 'Szwajcaria', 'CH'),  (5, 'Ukraina', 'UA'),  (6, 'Białoruś', 'BY'),  (7, 'Czarnogóra', 'MNE'),  (8, 'Wielka Brytania', 'GB'),  (9, 'Stany Zjednoczone', 'USA'),  (10, 'Kanada', 'CA');  -- --------------------------------------------------------  --  -- Stand-in structure for view `full\_address\_data`  -- (See below for the actual view)  --  CREATE TABLE `full\_address\_data` (  `first\_name` varchar(50)  ,`last\_name` varchar(100)  ,`street` varchar(200)  ,`house\_nr` varchar(10)  ,`post\_code` varchar(10)  ,`city` varchar(100)  ,`long\_name` varchar(200)  );  -- --------------------------------------------------------  --  -- Table structure for table `influencer`  --  CREATE TABLE `influencer` (  `Id\_influencer` int(11) NOT NULL,  `Address\_Id\_address` int(11) DEFAULT NULL,  `first\_name` varchar(50) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `last\_name` varchar(100) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `nickname` varchar(20) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `discount\_code` varchar(15) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `followers` int(11) DEFAULT NULL,  `account\_number` varchar(26) COLLATE utf8\_unicode\_ci DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `influencer`  --  INSERT INTO `influencer` (`Id\_influencer`, `Address\_Id\_address`, `first\_name`, `last\_name`, `nickname`, `discount\_code`, `followers`, `account\_number`) VALUES  (1, 4, 'Ismenka', 'Stelmaszczyk', 'ismena\_stelmaszczyk', 'ismena15', 47600, NULL),  (2, 2, 'Paulina', 'Guzińska', 'paulina\_guzinska', 'paulina15', 264000, NULL),  (3, 5, 'Bartosz', 'Smęda', 'smeda.triathlon', 'triathlon15', 801, NULL),  (4, 3, 'John', 'Smith', 'john\_smith', 'john15', 45000, '12445588237595145212547719'),  (5, 1, 'Maria', 'Wrześniak', 'mariaaa', 'marysia15', 53000, NULL),  (6, 6, 'Klaudia', 'Michalak', 'klaudia\_michalak', 'klaudia15', 4123, NULL),  (7, 7, 'Aleksander', 'Mąkosa', 'alexi', 'alexi15', 89000, NULL),  (8, 8, 'Aleksa', 'Woźnicki', 'aleksa\_woznikci', 'woznikci15', 632, NULL),  (9, 9, 'Rafał', 'Piotrowski', 'rafal\_p', 'rafal15', 7522, NULL),  (10, 10, 'Marcin', 'Wiesiuk', 'marcinek\_w', 'marcinek15', 65002, NULL);  -- --------------------------------------------------------  --  -- Table structure for table `influenceraction`  --  CREATE TABLE `influenceraction` (  `Id\_influencer\_action` int(11) NOT NULL,  `Influencer\_Id\_influencer` int(11) DEFAULT NULL,  `Service\_Id\_service` int(11) DEFAULT NULL,  `date` date DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `influenceraction`  --  INSERT INTO `influenceraction` (`Id\_influencer\_action`, `Influencer\_Id\_influencer`, `Service\_Id\_service`, `date`) VALUES  (1, 1, 8, '2021-09-30'),  (2, 1, 8, '2021-09-30'),  (3, 1, 1, '2021-09-15'),  (4, 2, 7, '2021-10-13'),  (5, 4, 10, '2021-11-02'),  (6, 3, 10, '2021-11-14'),  (7, 5, 3, '2021-10-19'),  (8, 7, 5, '2021-11-02'),  (9, 10, 9, '2021-11-03'),  (10, 8, 6, '2021-08-10'),  (11, 1, 1, '2021-12-01'),  (13, 4, 4, '2021-12-04'),  (14, 1, 9, '2022-01-13'),  (15, 4, 1, '2022-01-01'),  (16, 4, 4, '2022-01-04'),  (17, 2, 8, NULL);  --  -- Triggers `influenceraction`  --  DELIMITER $$  CREATE TRIGGER `new\_salary` AFTER INSERT ON `influenceraction` FOR EACH ROW INSERT INTO `salary` (`Id\_salary`, `Influencer\_Id\_influencer`, `InfluencerAction\_Id\_influencer\_action`, `value`, `date`) VALUES (NULL, NEW.Influencer\_Id\_influencer, NEW.Id\_influencer\_action,(SELECT value FROM `service` WHERE Id\_service = NEW.Service\_Id\_service), DATE(NOW()))  $$  DELIMITER ;  -- --------------------------------------------------------  --  -- Table structure for table `product`  --  CREATE TABLE `product` (  `Id\_product` int(11) NOT NULL,  `name` varchar(100) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `price` decimal(6,2) DEFAULT NULL,  `Amount` int(11) DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `product`  --  INSERT INTO `product` (`Id\_product`, `name`, `price`, `Amount`) VALUES  (1, 'Modify Reductor', '129.00', 94),  (2, 'Modify Femibra', '139.00', 67),  (3, 'Gumka do włosów - lniana ', '29.00', 20),  (4, 'Gumka do włosów - lniana XL', '49.00', 30),  (5, 'Gumka do włosów - welurowa', '19.00', 20),  (6, 'Gumka do włosów - welurowa XL', '29.00', 20),  (7, 'Pełna kuracja Modify Reductor', '387.00', 10),  (8, 'Pełna kuracja Modify Femibra', '417.00', 10),  (9, 'Zestaw świąteczny Modify Reductor', '199.00', 15),  (10, 'Zestaw świąteczny Modify Femibra', '256.00', 1),  (14, 'Nowy produkt dla User2', '100.00', 5);  -- --------------------------------------------------------  --  -- Table structure for table `salary`  --  CREATE TABLE `salary` (  `Id\_salary` int(11) NOT NULL,  `Influencer\_Id\_influencer` int(11) DEFAULT NULL,  `InfluencerAction\_Id\_influencer\_action` int(11) DEFAULT NULL,  `value` decimal(6,2) DEFAULT NULL,  `date` date DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `salary`  --  INSERT INTO `salary` (`Id\_salary`, `Influencer\_Id\_influencer`, `InfluencerAction\_Id\_influencer\_action`, `value`, `date`) VALUES  (1, 1, 1, '30.00', '2021-09-30'),  (2, 1, 2, '30.00', '2021-09-30'),  (3, 1, 3, '50.00', '2021-09-15'),  (4, 2, 4, '100.00', '2021-10-13'),  (5, 4, 5, '350.00', '2021-11-12'),  (6, 3, 6, '350.00', '2021-11-14'),  (7, 5, 7, '100.00', '2021-10-19'),  (8, 7, 8, '500.00', '2021-11-02'),  (9, 10, 9, '40.00', '2021-11-03'),  (10, 8, 10, '200.00', '2021-08-10'),  (13, 4, 13, '250.00', '2021-12-04'),  (14, 1, 14, '40.00', '2022-01-13'),  (15, 4, 15, '50.00', '2022-01-13'),  (16, 4, 16, '250.00', '2022-01-13'),  (17, 2, 17, '30.00', '2022-01-20');  -- --------------------------------------------------------  --  -- Table structure for table `service`  --  CREATE TABLE `service` (  `Id\_service` int(11) NOT NULL,  `name` varchar(255) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `value` decimal(6,2) DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `service`  --  INSERT INTO `service` (`Id\_service`, `name`, `value`) VALUES  (1, 'Relacja ', '50.00'),  (2, 'Relacja + swipe', '150.00'),  (3, 'Post bez produktu', '100.00'),  (4, 'Post z produktem', '250.00'),  (5, 'Udział w evencie marki', '500.00'),  (6, 'Logo marki na stronie', '200.00'),  (7, 'Oznaczenie na story', '100.00'),  (8, 'Użycie kodu influencera', '30.00'),  (9, 'Polecenie zakupu bez kodu', '40.00'),  (10, 'Pakiet (post + relacja + swipe)', '350.00');  -- --------------------------------------------------------  --  -- Table structure for table `shipping`  --  CREATE TABLE `shipping` (  `Id\_shipping` int(11) NOT NULL,  `Influencer\_Id\_influencer` int(11) DEFAULT NULL,  `Product\_Id\_product` int(11) DEFAULT NULL,  `date` date DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;  --  -- Dumping data for table `shipping`  --  INSERT INTO `shipping` (`Id\_shipping`, `Influencer\_Id\_influencer`, `Product\_Id\_product`, `date`) VALUES  (1, 1, 7, '2021-09-08'),  (2, 2, 7, '2021-11-07'),  (3, 3, 9, '2021-08-18'),  (4, 4, 1, '2021-11-19'),  (5, 5, 2, '2021-10-18'),  (6, 5, 5, '2021-10-18'),  (7, 6, 10, '2020-12-12'),  (8, 7, 9, '2021-11-17'),  (9, 8, 2, '2021-07-11'),  (10, 9, 1, '2021-11-06'),  (11, 10, 9, '2021-11-15'),  (12, 1, 4, '2021-11-08'),  (13, 6, 2, '2022-01-20');  --  -- Triggers `shipping`  --  DELIMITER $$  CREATE TRIGGER `update\_stock` AFTER INSERT ON `shipping` FOR EACH ROW BEGIN  CALL ProductStock (NEW.Product\_Id\_product, 1);  END  $$  DELIMITER ;  -- --------------------------------------------------------  --  -- Structure for view `full\_address\_data`  --  DROP TABLE IF EXISTS `full\_address\_data`;  CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `full\_address\_data` AS SELECT `i`.`first\_name` AS `first\_name`, `i`.`last\_name` AS `last\_name`, `a`.`street` AS `street`, `a`.`house\_nr` AS `house\_nr`, `a`.`post\_code` AS `post\_code`, `a`.`city` AS `city`, `c`.`long\_name` AS `long\_name` FROM ((`influencer` `i` left join `address` `a` on(`i`.`Address\_Id\_address` = `a`.`Id\_address`)) left join `country` `c` on(`a`.`Country\_Id\_country` = `c`.`Id\_country`)) ;  --  -- Indexes for dumped tables  --  --  -- Indexes for table `address`  --  ALTER TABLE `address`  ADD PRIMARY KEY (`Id\_address`),  ADD KEY `Country\_Id\_country` (`Country\_Id\_country`);  --  -- Indexes for table `country`  --  ALTER TABLE `country`  ADD PRIMARY KEY (`Id\_country`);  --  -- Indexes for table `influencer`  --  ALTER TABLE `influencer`  ADD PRIMARY KEY (`Id\_influencer`),  ADD KEY `Address\_Id\_address` (`Address\_Id\_address`);  --  -- Indexes for table `influenceraction`  --  ALTER TABLE `influenceraction`  ADD PRIMARY KEY (`Id\_influencer\_action`),  ADD KEY `Influencer\_Id\_influencer` (`Influencer\_Id\_influencer`),  ADD KEY `Service\_Id\_service` (`Service\_Id\_service`);  --  -- Indexes for table `product`  --  ALTER TABLE `product`  ADD PRIMARY KEY (`Id\_product`);  --  -- Indexes for table `salary`  --  ALTER TABLE `salary`  ADD PRIMARY KEY (`Id\_salary`),  ADD KEY `InfluencerAction\_Id\_influencer\_action` (`InfluencerAction\_Id\_influencer\_action`),  ADD KEY `Influencer\_Id\_influencer` (`Influencer\_Id\_influencer`);  --  -- Indexes for table `service`  --  ALTER TABLE `service`  ADD PRIMARY KEY (`Id\_service`);  --  -- Indexes for table `shipping`  --  ALTER TABLE `shipping`  ADD PRIMARY KEY (`Id\_shipping`),  ADD KEY `Influencer\_Id\_influencer` (`Influencer\_Id\_influencer`),  ADD KEY `Product\_Id\_product` (`Product\_Id\_product`);  --  -- AUTO\_INCREMENT for dumped tables  --  --  -- AUTO\_INCREMENT for table `address`  --  ALTER TABLE `address`  MODIFY `Id\_address` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=12;  --  -- AUTO\_INCREMENT for table `country`  --  ALTER TABLE `country`  MODIFY `Id\_country` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=11;  --  -- AUTO\_INCREMENT for table `influencer`  --  ALTER TABLE `influencer`  MODIFY `Id\_influencer` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=11;  --  -- AUTO\_INCREMENT for table `influenceraction`  --  ALTER TABLE `influenceraction`  MODIFY `Id\_influencer\_action` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=18;  --  -- AUTO\_INCREMENT for table `product`  --  ALTER TABLE `product`  MODIFY `Id\_product` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=15;  --  -- AUTO\_INCREMENT for table `salary`  --  ALTER TABLE `salary`  MODIFY `Id\_salary` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=18;  --  -- AUTO\_INCREMENT for table `service`  --  ALTER TABLE `service`  MODIFY `Id\_service` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=11;  --  -- AUTO\_INCREMENT for table `shipping`  --  ALTER TABLE `shipping`  MODIFY `Id\_shipping` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=14;  --  -- Constraints for dumped tables  --  --  -- Constraints for table `address`  --  ALTER TABLE `address`  ADD CONSTRAINT `address\_ibfk\_1` FOREIGN KEY (`Country\_Id\_country`) REFERENCES `country` (`Id\_country`);  --  -- Constraints for table `influencer`  --  ALTER TABLE `influencer`  ADD CONSTRAINT `influencer\_ibfk\_1` FOREIGN KEY (`Address\_Id\_address`) REFERENCES `address` (`Id\_address`);  --  -- Constraints for table `influenceraction`  --  ALTER TABLE `influenceraction`  ADD CONSTRAINT `influenceraction\_ibfk\_1` FOREIGN KEY (`Influencer\_Id\_influencer`) REFERENCES `influencer` (`Id\_influencer`),  ADD CONSTRAINT `influenceraction\_ibfk\_2` FOREIGN KEY (`Service\_Id\_service`) REFERENCES `service` (`Id\_service`);  --  -- Constraints for table `salary`  --  ALTER TABLE `salary`  ADD CONSTRAINT `salary\_ibfk\_1` FOREIGN KEY (`InfluencerAction\_Id\_influencer\_action`) REFERENCES `influenceraction` (`Id\_influencer\_action`),  ADD CONSTRAINT `salary\_ibfk\_2` FOREIGN KEY (`Influencer\_Id\_influencer`) REFERENCES `influencer` (`Id\_influencer`);  --  -- Constraints for table `shipping`  --  ALTER TABLE `shipping`  ADD CONSTRAINT `shipping\_ibfk\_1` FOREIGN KEY (`Influencer\_Id\_influencer`) REFERENCES `influencer` (`Id\_influencer`),  ADD CONSTRAINT `shipping\_ibfk\_2` FOREIGN KEY (`Product\_Id\_product`) REFERENCES `product` (`Id\_product`);  COMMIT;  /\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  /\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  /\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/; |
| --- |