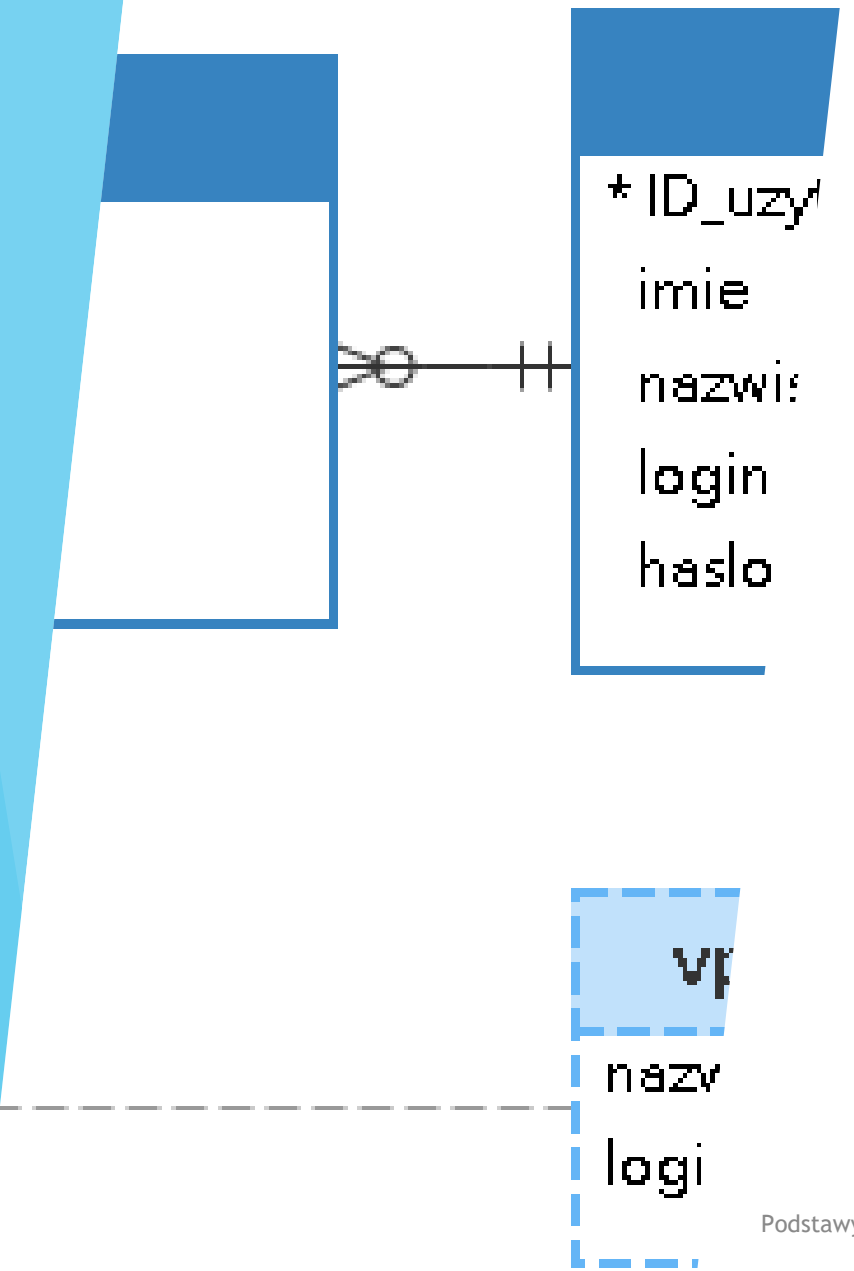
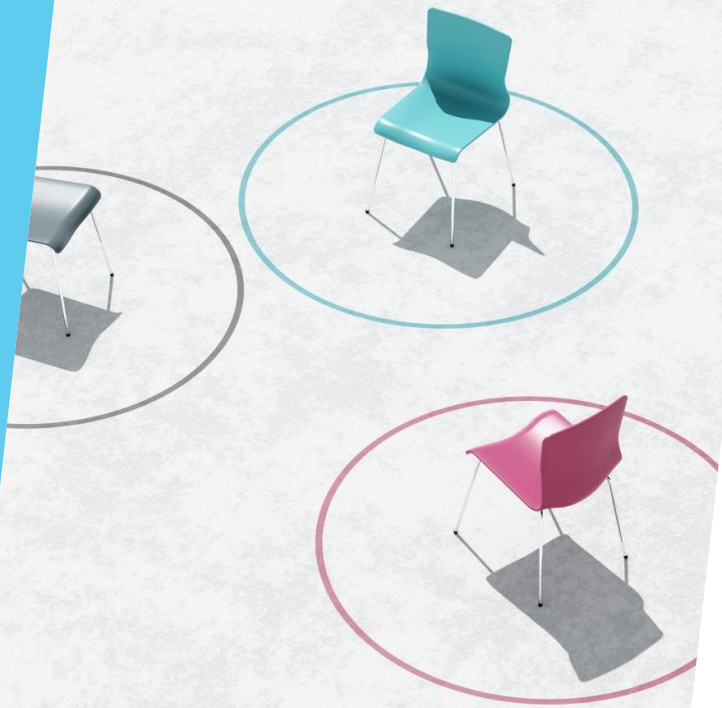



# Projektujemy własną bazę danych (schemat)



# Odpowiedzi na następujące pytania:

- Jak zaprojektować schemat bazy danych? Diagram ERD.
- Jak budować poprawnie tabele?
- Jakie mamy typy relacji pomiędzy tabelami?
- Czym są postacie normalne w bazach danych?
- Co to jest nadmiarowość danych i jak jej unikać?
- Co to są anomalie: modyfikacji, usunięć i wstawiania?
- Jakich typów danych używać?
- Jakich indeksów używać w tabelach?
- Czy (kiedy) partycjonować tabele?
- Jakich silników używać w tabelach?
- Czy zabezpieczyć dane w postaci wersjonowania?

# Jak zaprojektować schemat bazy danych?

## (w relacyjnym systemie zarządzania bazą danych)

# Jak poprawnie budować tabele?

- Tabele w bazie danych należy budować z zachowaniem pewnych norm, dzięki którym baza będzie spójna, czytelna, a dane nie będą zajmowały zbędnego miejsca na dysku.

# Jak projektować schemat bazy danych?

- **Normalizacja baz danych** jest procesem mającym na celu zlikwidowanie danych niepotrzebnie powtarzających się w relacyjnej bazie danych.

- Dzięki normalizacji zmniejsza się ryzyko powstawania:
  - nadmiarowości danych – redundancja (w tym przypadku niepożądana),
  - anomalii modyfikacji (aktualizacji),
  - anomalii usunięć oraz
  - anomalii wstawiania.

**Nadmiarowość danych** ma miejsce wtedy, kiedy dana wartość niepotrzebnie jest powielana w krotkach danego atrybutu (kolumny). Kiedy np. podczas dodawania kolejnego wiersza, użytkownik popełni błąd w wartości powielanej danej, to baza traci spójność – **anomalia wstawiania.**

```
CREATE TABLE uzytkownicy_programu_pocztowego (
  ID_upp int(10) unsigned NOT NULL AUTO_INCREMENT,
  imie varchar(50) NOT NULL,
  nazwisko varchar(255) NOT NULL,
  login varchar(50) NOT NULL,
  haslo varchar(255) NOT NULL,
  adres_wysylki_miejscowosc varchar(100) DEFAULT NULL,
  adres_wysylki_kraj varchar(100) DEFAULT NULL,
  adres_wysylki_ulica varchar(100) DEFAULT NULL,
  PRIMARY KEY (ID_upp),
  UNIQUE KEY login (login)
)
```

| ID_upp | imie | nazwisko | login | haslo | adres_wysylki_kraj | adres_wysylki_ulica |
|---|---|---|---|---|---|---|
| 1 | Bernard | King | befis | *2AE8F37A5A526EF3C5AD82: | USA | 762 Riverview Road |
| 2 | Judy | Harrison | judy1 | *7F0CEF3DBCDC5CD9B3B69[ | USA | 279 Regent Street |
| 3 | Wing Kuer | Kim | wingkuenpan | *1738CE2672AC1C3E8A4BFFF | Japonia | 282 Sherman E Rd, Cai |
| 4 | Ka Keung | Wallace | kakeungmak | *7D849AD352D13EA0FAB783 | Japonia | 406 Dong Zhi Men, Dor |
| 5 | Kwok Yin | Silva | kwokyintong06 | *E7FB107AD2E8A597B7AA1C | USA | 825 1st Ave |
| 6 | Tony | Walker | tony4 | *4BDCFC8F115AB7F99F0584F | U S A | 861 Whitehouse Lane, I |
| 7 | Russell | Perez | russell1952 | *17A89442F5A163782E2262B4 | Stanu Zjednoczone | 308 Lark Street |
| 8 | Dawn | Rogers | dawng | *39A87BA0E524D1A991E26A6 | USA | 356 Wooster Street |
| 9 | Cindy | Young | pacindy | *2D9CAB2513206F837A9B579 | USA | 363 Nostrand Ave |

**Anomalia modyfikacji** – chęć zmiany powielanej informacji wymusza zmianę tej informacji we wszystkich krotkach. Pominięcie choćby jednej krotki powoduje utratę spójności danych.

Uaktualnianie powielanej danej zabiera niepotrzebnie dużo czasu i zajmuje proces.

**Chcemy uaktualnić nazwę miejscowości „Krynica Zdrój” na „Krynica-Zdrój”**

| ID_uzytkownik | login | imie | nazwisko | miejscowosc_urodzenia | miejscowosc_pobytu |
|---|---|---|---|---|---|
| 1 | jan | Jan | Kowalski | Krynica Zdrój | Krynica Zdrój |
| 2 | anna | Anna | Nowak | Krynica Zdrój | Kraków |
| 3 | michal | Michał | Niemczyk | Krynica Zdrój | Nowy Sącz |
| 4 | janina | Janina | Luty | Krynica Zdruj | Zakopane |

| ID_produkty | nazwa | producent_adres_miejscowc |
|---|---|---|
| 1 | R45 | Krynica Zdrój |
| 2 | X990 | Kraków |
| 3 | Z80 | Krynica Zdrój |
| 4 | W234 | Krynica_Zdrój |

**Anomalia usunięć** – ma miejsce wtedy, gdy usuniemy wiersz, który przechowywał istotną informację, która w konsekwencji została niepotrzebnie utracona.

| ID_uzytkownik | login | imie | nazwisko | miejscowosc_pobytu_naszego_klienta |
|---|---|---|---|---|
| 1 | jan | Jan | Kowalski | Krynica-Zdrój |
| 2 | anna | Anna | Nowak | Warszawa |
| 3 | michal | Michał | Niemczyk | Kraków |
| 4 | janina | Janina | Luty | Krynica-Zdrój |

# Kiedy powielane informacje nie są wynikiem błędnego projektu?

Przykład własnego mechanizmu historii rekordów z wykorzystaniem wyzwalaczy

```
CREATE TABLE `uzytkownicy` (
  `ID_uzytkownik` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `login` varchar(20) NOT NULL,
  `imie` varchar(20) NOT NULL,
  `nazwisko` varchar(80) NOT NULL,
  `haslo` varchar(32) NOT NULL,
  PRIMARY KEY (`ID_uzytkownik`),
  UNIQUE KEY `uniq_login` (`login`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;

CREATE TRIGGER `archiwum_uzytkownicy_update` AFTER UPDATE ON `uzytkownicy` FOR EACH ROW insert into archiwum_uzytkownicy
(ID_uzytkownik, login, imie, nazwisko, haslo, akcja) values
(old.ID_uzytkownik, old.login, old.imie, old.nazwisko, old.haslo, 'u');

CREATE TRIGGER `archiwum_uzytkownicy_delete` AFTER DELETE ON `uzytkownicy` FOR EACH ROW insert into archiwum_uzytkownicy
(ID_uzytkownik, login, imie, nazwisko, haslo, akcja) values
(old.ID_uzytkownik, old.login, old.imie, old.nazwisko, old.haslo, 'd');
```

```
CREATE TABLE `archiwum_uzytkownicy` (
  `ID_archiwum_uzytkownicy` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `ID_uzytkownik` int(11) NOT NULL,
  `login` varchar(20) NOT NULL,
  `imie` varchar(20) NOT NULL,
  `nazwisko` varchar(80) NOT NULL,
  `haslo` varchar(32) DEFAULT NULL,
  `akcja` char(1) NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`ID_archiwum_uzytkownicy`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

# Kiedy powielane informacje nie są wynikiem błędnego projektu?

| ID_uzytkownik | login | imie | nazwisko | haslo |
|---|---|---|---|---|
| 1 | jan | Janek | Kowalski | 202cb962a |
| 2 | anna | Anna | Nowak | |
| 3 | michal | Michał | Niemczyk | |

| ID_archiwum_uzytkownicy | ID_uzytkownik | login | imie | nazwisko | haslo | akcja | timestamp |
|---|---|---|---|---|---|---|---|
| 1 | 1 | jan | Jan | Kowalski | | u | 2014-11-11 19:32:20 |
| 2 | 4 | janina | Janina | Luty | | d | 2014-11-11 19:32:30 |

# Jak projektować schemat bazy danych?

- **Denormalizacja baz danych** odnosi się do celowego zastosowania redundancji danych w celu poprawy wydajności odczytu i uproszczenia złożonych operacji zapytań

- Denormalizacja jest używana w sytuacjach, gdy priorytetem jest optymalizacja wydajności odczytu kosztem wydajności zapisu oraz kiedy zapytania są częściej wykonywane niż operacje zapisu.
- Przykłady takich sytuacji to: hurtownie danych, raportowanie, analiza danych czy aplikacje, w których konieczne jest szybkie dostarczanie wyników.

# Jak projektować schemat bazy danych?

Główne cechy denormalizacji to:

- Redundancja danych: denormalizacja wprowadza celową redundancję danych poprzez powielenie pewnych informacji w różnych miejscach w bazie danych.
- Poprawa wydajności odczytu: Denormalizacja umożliwia optymalizację operacji odczytu, ponieważ dane, które są często pobierane razem, są przechowywane blisko siebie. Eliminuje potrzebę łączenia wielu tabel podczas wykonywania złożonych zapytań, co przyspiesza dostęp do danych.

# Jak projektować schemat bazy danych?

- Uproszczenie operacji zapytań: Poprzez redundancję danych, denormalizacja może uprościć złożone operacje zapytań, eliminując potrzebę łączenia tabel i wykonując mniej operacji.
- Zwiększenie złożoności zapisu: Jednym z kosztów denormalizacji jest zwiększenie złożoności operacji zapisu. Ponieważ dane są replikowane w różnych miejscach, konieczne jest dbanie o ich spójność i aktualizację w każdym wystąpieniu.

# Jak projektować schemat bazy danych?

- Denormalizacja może prowadzić do trudniejszego zarządzania danymi, większej podatności na błędy i utraty spójności danych. Zastosowanie denormalizacji powinno być dobrze przemyślane i oparte na analizie wymagań i charakterystyki danych w konkretnym kontekście aplikacji.

# Prosty przepis na względnie poprawny projekt bazy danych w aspekcie normalizacji

- Wypunktować to co powinna przechowywać baza danych.
- Pogrupować w tabele powyższe dane tak, aby jednoznacznie charakteryzowały dany obiekt.
- Poszczególne kolumny (jeżeli wymaga tego kontekst projektu/problemu) powinny zawierać dane atomowe, np. kolumna **adres** powinna zostać rozbita na kolumny przechowujące: nazwę ulicy, numer domu, numer mieszkania, miasto, kod pocztowy itd.
- Każda tabela powinna zawierać kolumnę jednoznacznie identyfikującą wiersze w pozostałych kolumnach (tzw. klucz główny)

# Prosty przepis na względnie poprawny projekt bazy danych w aspekcie normalizacji

- Wyeliminować zależności przechodnie, czyli wykluczyć takie kolumny, których wartości nie są bezpośrednio zależne od klucza głównego, ale od innej
- Jeżeli jest taka możliwość, to połączyć kolumny w danej tabeli z kolumnami jednoznacznie identyfikującym wiersze w tabeli pokrewnej (referencje PK-FK).

# Poprawny projekt bazy danych w aspekcie normalizacji:

- Podczas normalizacji bazy danych dzieli się dane na mniejsze, bardziej spójne i znormalizowane tabele.
- Istnieje kilka poziomów normalizacji, które są znane też pod nazwą formy normalne (ang. *Normal Form*).
- Najbardziej powszechnie stosowane i najprostsze technicznie do implementacji to: pierwsza, druga i trzecia postać normalna (1NF, 2NF, 3NF)
- Istnieją również wyższe poziomy normalizacji, takie jak czwarta i piąta postać normalna (4NF, 5NF) oraz postacie normalne Boyce'a-Codda (BCNF) i szósta postać normalna (6NF).

# Poprawny projekt bazy danych w aspekcie normalizacji:

- Normalizacja jest ważnym procesem w projektowaniu baz danych, ponieważ prowadzi do bardziej efektywnych i spójnych struktur danych, ułatwiających zarządzanie i przetwarzanie informacji.
- Umożliwia także unikanie problemów związanych z anomaliami danych, takich jak utrata spójności lub trudności w aktualizacji, co przyczynia się do większej integralności danych w bazie danych.

# Postacie normalne

# Kilka ważnych pojęć

**Klucz główny (ang. *primary key*)** - to pojęcie w relacyjnych bazach danych odnoszące się do zestawu jednej lub więcej kolumn w tabeli, które razem zapewniają unikalność wiersza w tej tabeli. Innymi słowy, nie mogą istnieć dwa wiersze w tabeli z takim samym zestawem wartości w kolumnach klucza głównego.

Cechy klucza głównego:

- **Unikalność**: Wartości w kolumnie (lub kolumnach) klucza głównego muszą być unikalne dla każdego wiersza w tabeli.
- **Niezmienność**: Raz przypisana wartość klucza głównego nie powinna być modyfikowana (aczkolwiek technicznie może być). Jeśli wartość klucza głównego wiersza zostanie zmieniona, wiersz ten staje się, z punktu widzenia bazy danych, zupełnie nowym wierszem.
- **Nie może być pusty**: Kolumny klucza głównego nie mogą mieć wartości NULL. Oznacza to, że musi istnieć wartość klucza głównego dla każdego wiersza w tabeli.

# Kilka ważnych pojęć

Cechy klucza głównego:

- **Integralność**: Klucz główny pomaga w utrzymaniu integralności danych w bazie. Jeśli kolumna w innej tabeli jest zdefiniowana jako klucz obcy, który odnosi się do klucza głównego w danej tabeli, system bazodanowy zapewni, że każda wartość klucza obcego odpowiada istniejącej wartości klucza głównego.
- **Może być złożony**: Klucz główny może składać się z jednej lub więcej kolumn. Jeśli klucz główny składa się z więcej niż jednej kolumny, nazywa się to kluczem złożonym.
- W tabeli może istnieć wyłącznie **jeden klucz główny**. System bazodanowy nie pozwoli na utworzenie większej liczby kluczy głównych.

# Kilka ważnych pojęć

**Klucz kandydujący (ang. *candidate key*)** – to kolumna lub zestaw kolumn w tabeli bazy danych, które mogą być używane jako klucz główny. Innymi słowy, klucz kandydujący jest unikalny dla każdego wiersza w tabeli i nie zawiera wartości NULL.

Cechy klucza kandydującego:

- **Unikalność:** Podobnie jak klucz główny, klucz kandydujący zapewnia unikalność wiersza w tabeli. Oznacza to, że nie ma dwóch wierszy w tabeli z takim samym zestawem wartości dla klucza kandydującego.
- **Brak wartości NULL**: Klucz kandydujący nie może zawierać wartości NULL.
- **Istnienie wielu kluczy kandydujących**: W jednej tabeli może istnieć wiele kluczy kandydujących. Spośród tych kluczy kandydujących można wybrać jeden jako klucz główny, a pozostałe staną się alternatywnymi kluczami kandydującymi.

# Kilka ważnych pojęć

**Przykład:**

Załóżmy, że mamy tabelę *Studenci* z kolumnami:
*Numer_Indeksu*, *PESEL* i *Email*.

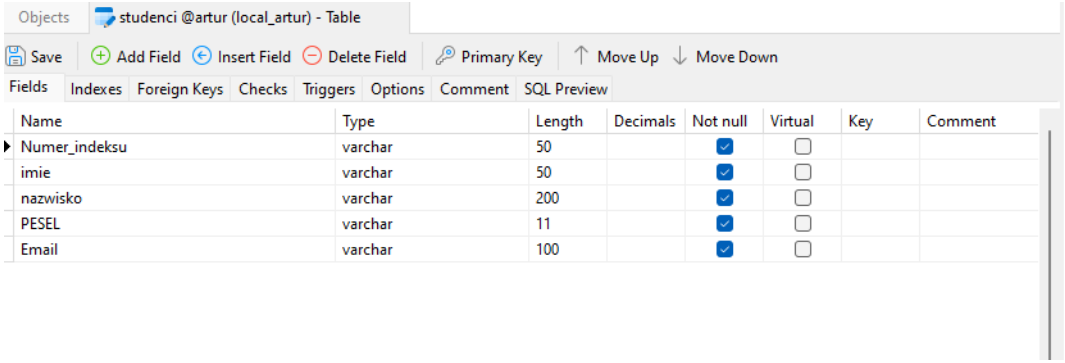
Jeśli:

- każdy student ma unikalny *Numer_Indeksu*, unikalny *PESEL* i unikalny *Email*,
- żadne z tych pól nie może być *NULL*,

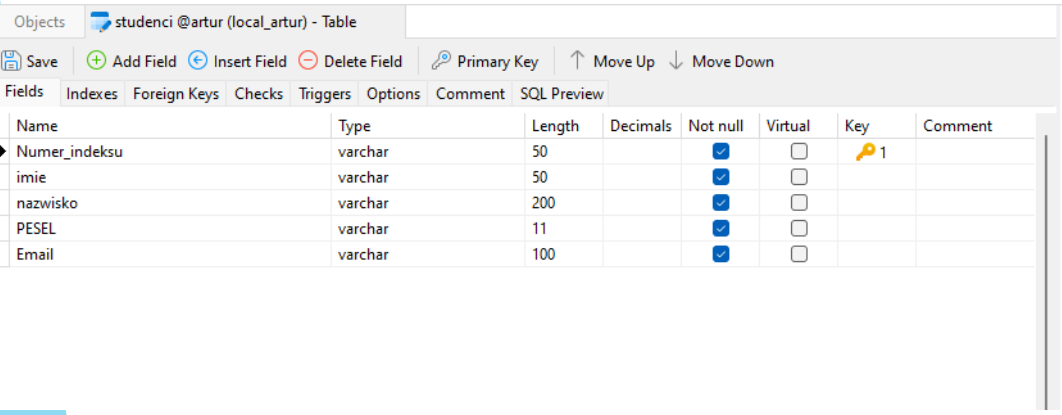
to wszystkie trzy kolumny (*Numer_Indeksu*, *PESEL*, *Email*) są **kluczami kandydującymi**, ponieważ każda z nich może służyć jako klucz główny do identyfikacji studenta w tabeli.

Jednak w praktyce zazwyczaj wybieramy jeden z tych kluczy kandydujących jako **klucz główny** (np. *Numer_Indeksu*), podczas gdy pozostałe dwie kolumny pozostają **alternatywnymi kluczami kandydującymi**.

# Kilka ważnych pojęć




```
CREATE TABLE `studenci` (
  `Numer_indeksu` varchar(50) NOT NULL,
  `imie` varchar(50) NOT NULL,
  `nazwisko` varchar(200) NOT NULL,
  `PESEL` varchar(11) NOT NULL,
  `Email` varchar(100) NOT NULL,
  UNIQUE KEY `Numer_indeksu` (`Numer_indeksu`) USING BTREE,
  UNIQUE KEY `PESEL` (`PESEL`) USING BTREE,
  UNIQUE KEY `Email` (`Email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```




```
CREATE TABLE `studenci` (
  `Numer_indeksu` varchar(50) NOT NULL,
  `imie` varchar(50) NOT NULL,
  `nazwisko` varchar(200) NOT NULL,
  `PESEL` varchar(11) NOT NULL,
  `Email` varchar(100) NOT NULL,
  PRIMARY KEY (`Numer_indeksu`),
  UNIQUE KEY `Numer_indeksu` (`Numer_indeksu`) USING BTREE,
  UNIQUE KEY `PESEL` (`PESEL`) USING BTREE,
  UNIQUE KEY `Email` (`Email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

# Postacie normalne

W praktyce wiele baz danych jest normalizowanych do 3PN lub BCNF, ponieważ te poziomy zapewniają dobrą równowagę między zredukowaniem redundancji a zachowaniem wydajności i prostoty projektu. Niemniej jednak, w niektórych przypadkach, dalsza normalizacja może być wymagana w celu osiągnięcia konkretnych celów projektowych.

- **Pierwsza postać normalna (1PN):**
  - Każda tabela ma klucz główny: unikalny identyfikator dla każdego wiersza.
  - Wszystkie kolumny w tabeli przechowują jedną wartość logiczną - atomową (brak zbiorów, list czy innych struktur wieloelementowych).
  - Kolejność wierszy i kolumn jest nieistotna.

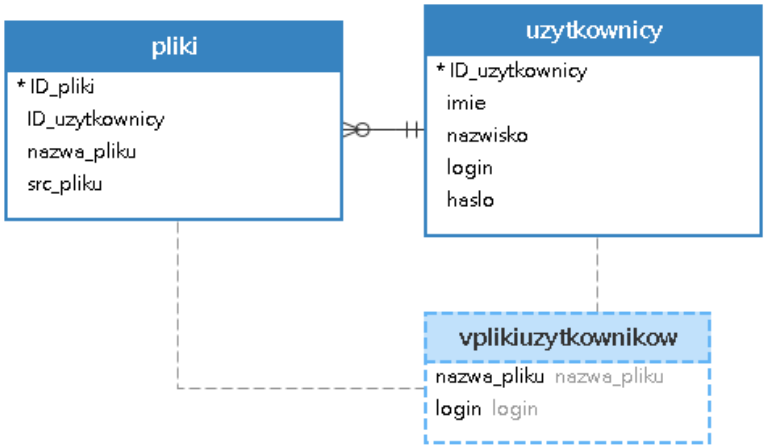
# Postacie normalne

- **Druga postać normalna (2PN):**
  - Tabela jest w 1PN.
  - Wszystkie niekluczowe kolumny są funkcjonalnie zależne od całego klucza głównego, a nie tylko od jego części (dotyczy to przede wszystkim tabel z kluczami złożonymi).

- **Trzecia postać normalna (3PN):**
  - Tabela jest w 2PN.
  - Wszystkie atrybuty są funkcjonalnie zależne tylko od klucza głównego, a nie od innych niekluczowych atrybutów.

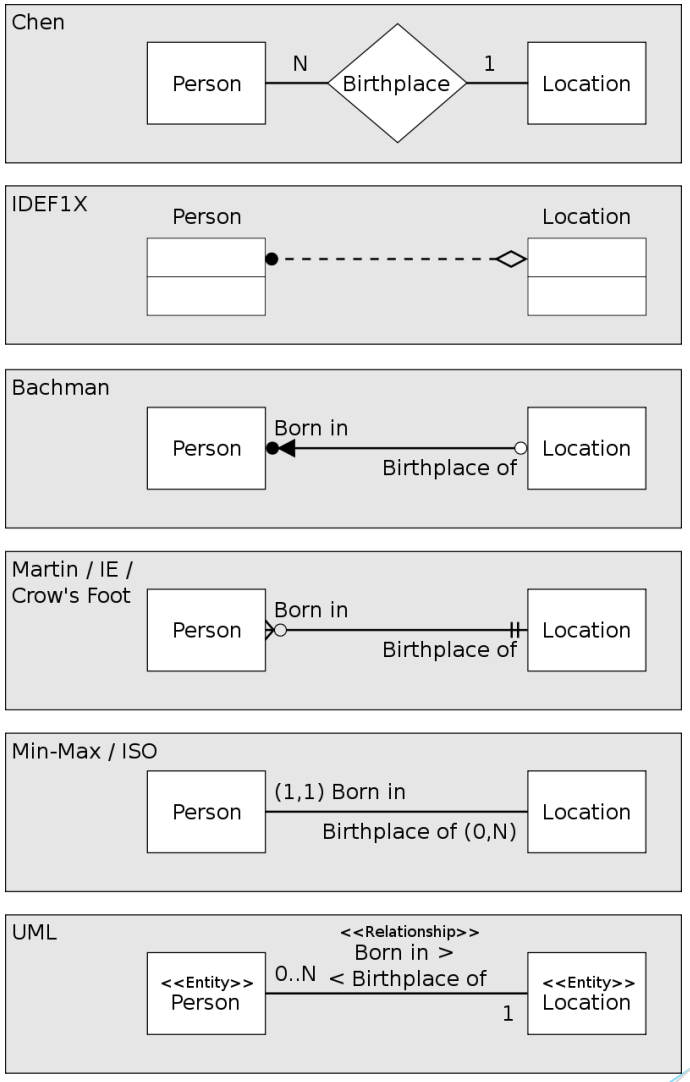
# Diagram ER (ERD)

- Diagram związków encji (ang. ***e**ntity-relationship **d**iagram*) – pozwala na graficzne zamodelowanie struktur danych oraz relacji pomiędzy nimi.



# Diagram ERD

- Przykładowe notacje:
  - Chen,
  - Bachmana,
  - Crows's Foot
  - Min-max (ISO)
  - UML



# Diagram ERD

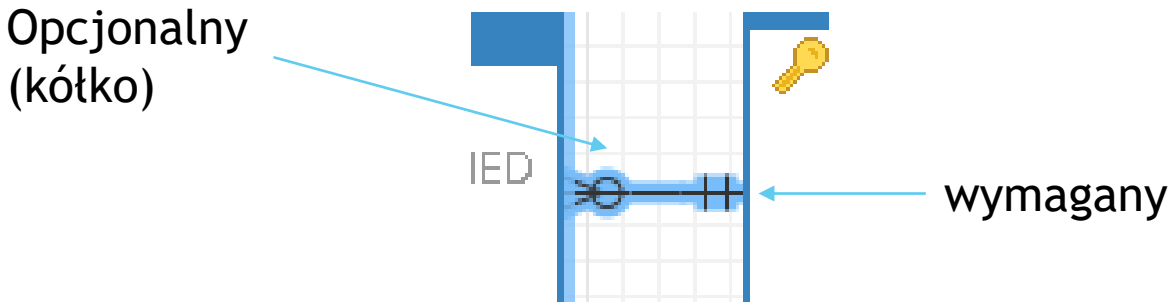
**Podstawowe pojęcia:**

- encja - reprezentacja obiektu ze świata rzeczywistego lub obiektu abstrakcyjnego – np. student, pracownik
- atrybut – element encji wchodzący w skład jej cech ją identyfikujących – np. imie, nazwisko, rok urodzenia
- związek - powiązanie między dwiema lub kilkoma encjami.

# Diagram ERD

Dwie główne cechy związku:

1. Opcjonalność – która mówi o tym, czy każda encja musi, czy też może wystąpić równocześnie z inną.




2. Krotność – określającą ile encji wchodzi w skład związku:
   1:1 („jeden do jeden”) – encji odpowiada dokładnie jedna encja,
   1:N („jeden do wielu”) – encji odpowiada jedna lub więcej encji,
   M:N („wiele do wielu”) – jednej lub więcej encjom odpowiada jedna lub więcej encji.

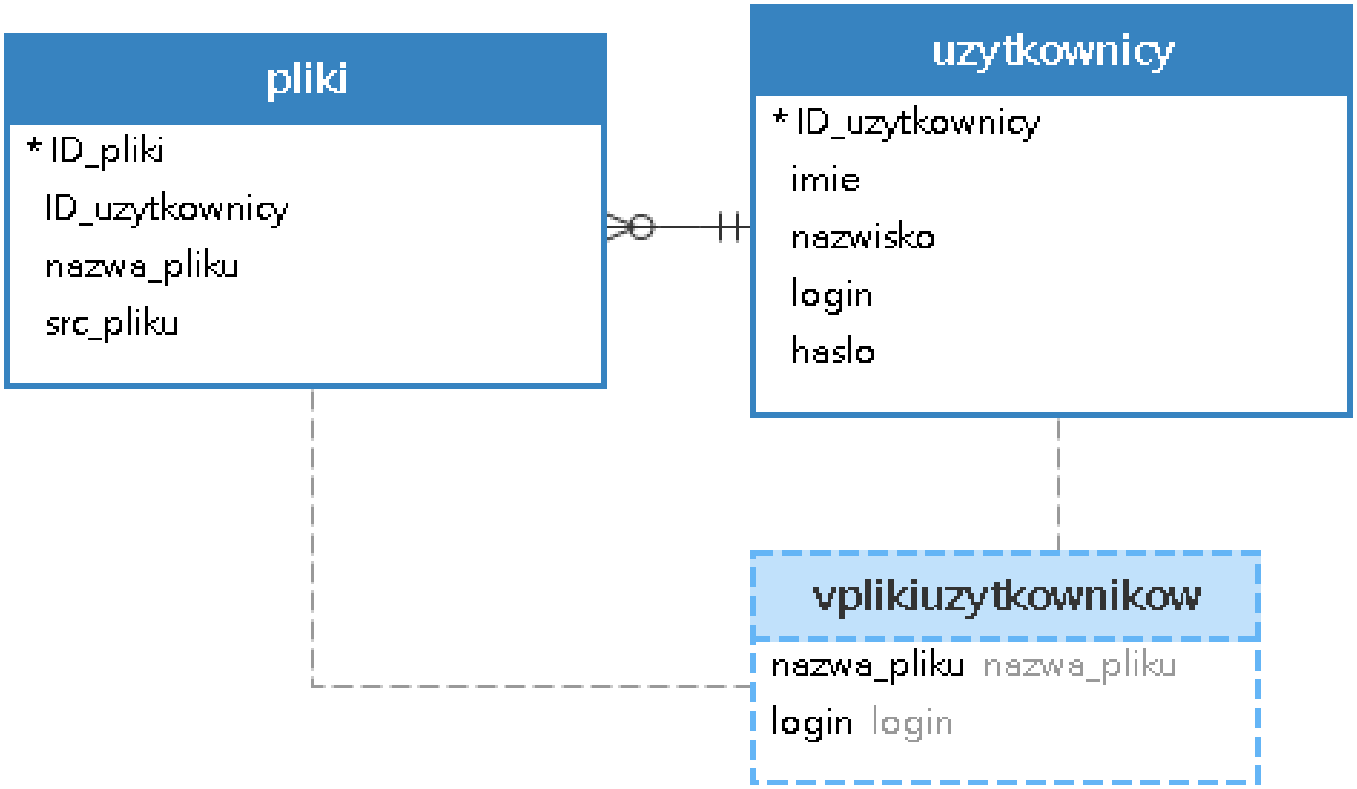
# Krotność = Relacje pomiędzy tabelami

- Relacjami pomiędzy tabelami nazywamy logiczne połączenia danych rozmieszonych w tych tabelach ostatecznie tworzące pewien rodzaj całości.

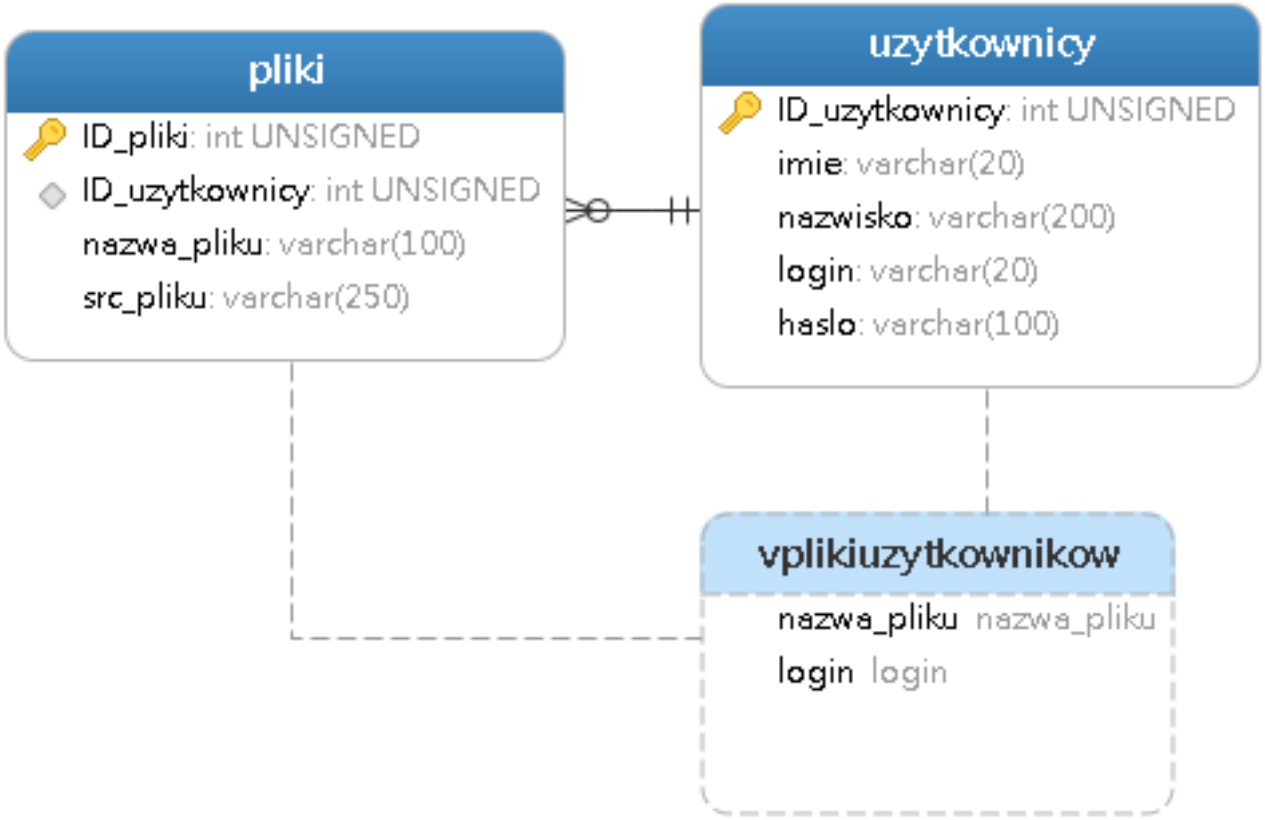
- Mamy następujące typy relacji:

  - **jeden-do-jednego** – relacja rzadko spotykana, tabela pierwsza może mieć tylko jeden rekord dopasowany z drugiej tabeli, i również druga tabela może mieć tylko jeden rekord dopasowany z tabeli pierwszej.
  - **jeden-do-wielu** – powszechny typ relacji. Rekord z tabeli pierwszej może mieć wiele dopasowań rekordów z tabeli drugiej, natomiast rekord z tabeli drugiej może mieć tylko jedno dopasowanie rekordu z tabeli pierwszej.
  - **wiele-do-wielu** - wiele rekordów z tabeli pierwszej może być dopasowanych do wielu rekordów z tabeli drugiej, i na odwrót. W rzeczywistości tworzona jest trzecia tabela, która odpowiednio tworzy most pomiędzy tabelami (tzw. tabela łącza). Tabela łącza posiada dwa klucze obce – z tabeli pierwszej i z tabeli drugiej.

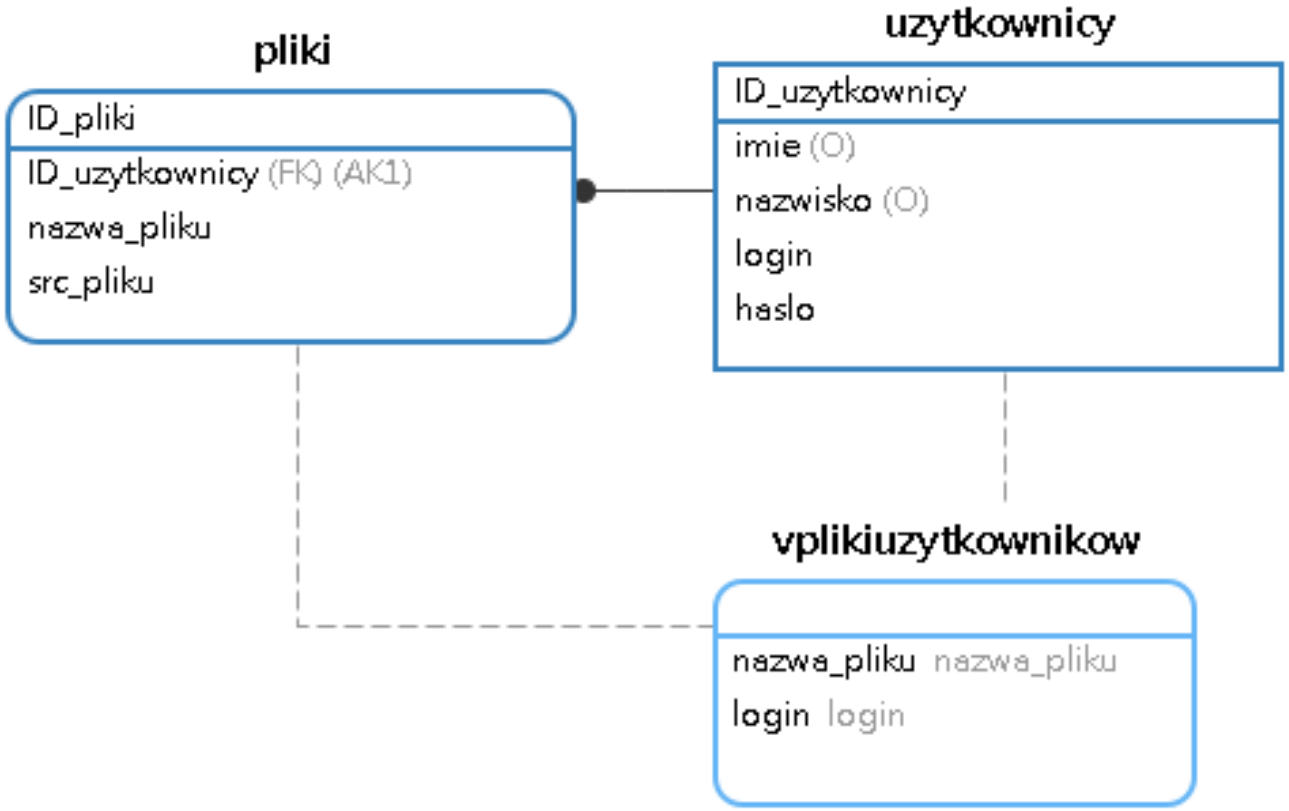
# Diagram ERD – notacja Crows's Foot (kurzej łapki)



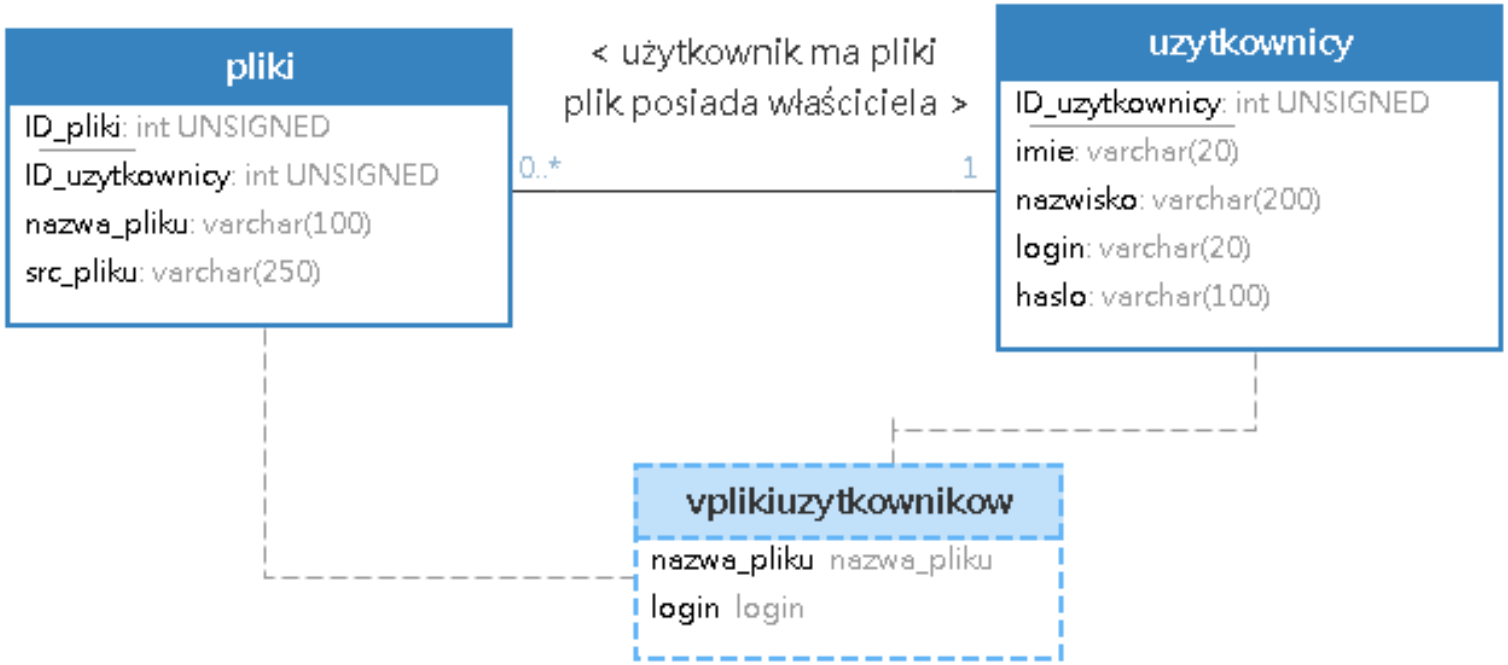
# Diagram ERD – notacja Crows's Foot (kurzej łapki)



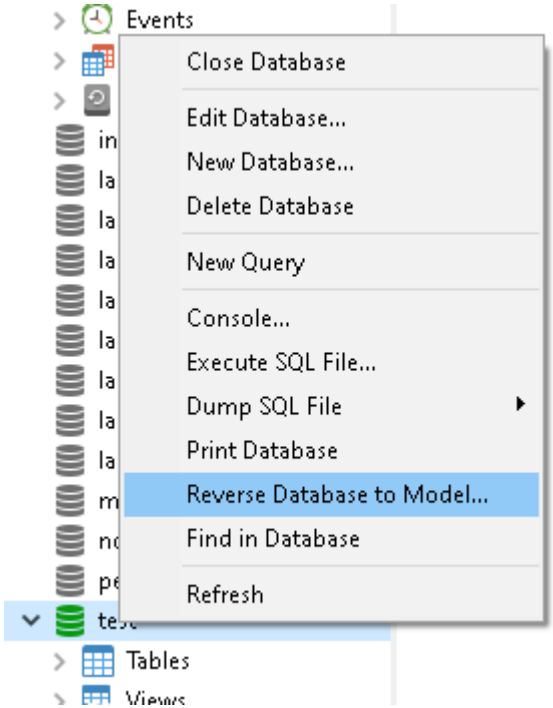
# Diagram ERD – notacja IDEF1X



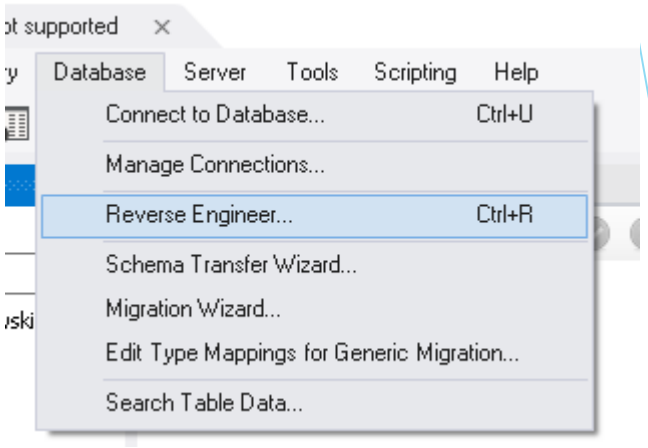
# Diagram ERD – notacja UML



# Diagramy ERD w programach desktopowych

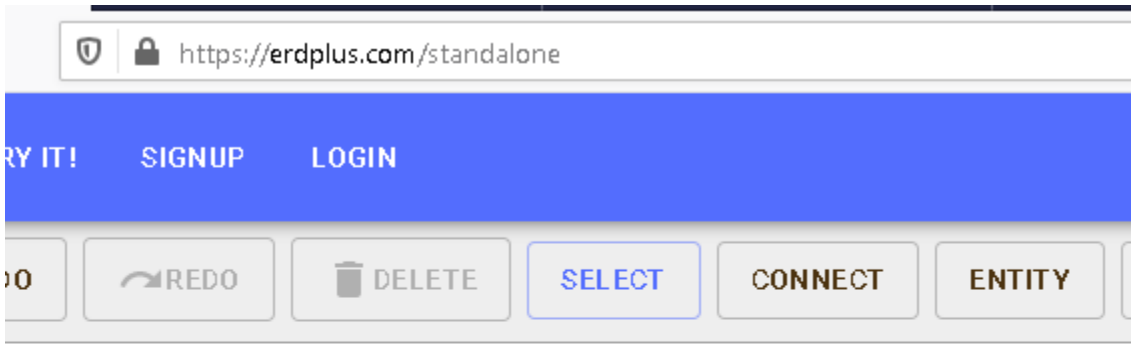



Navicat




MySQL
Workbench

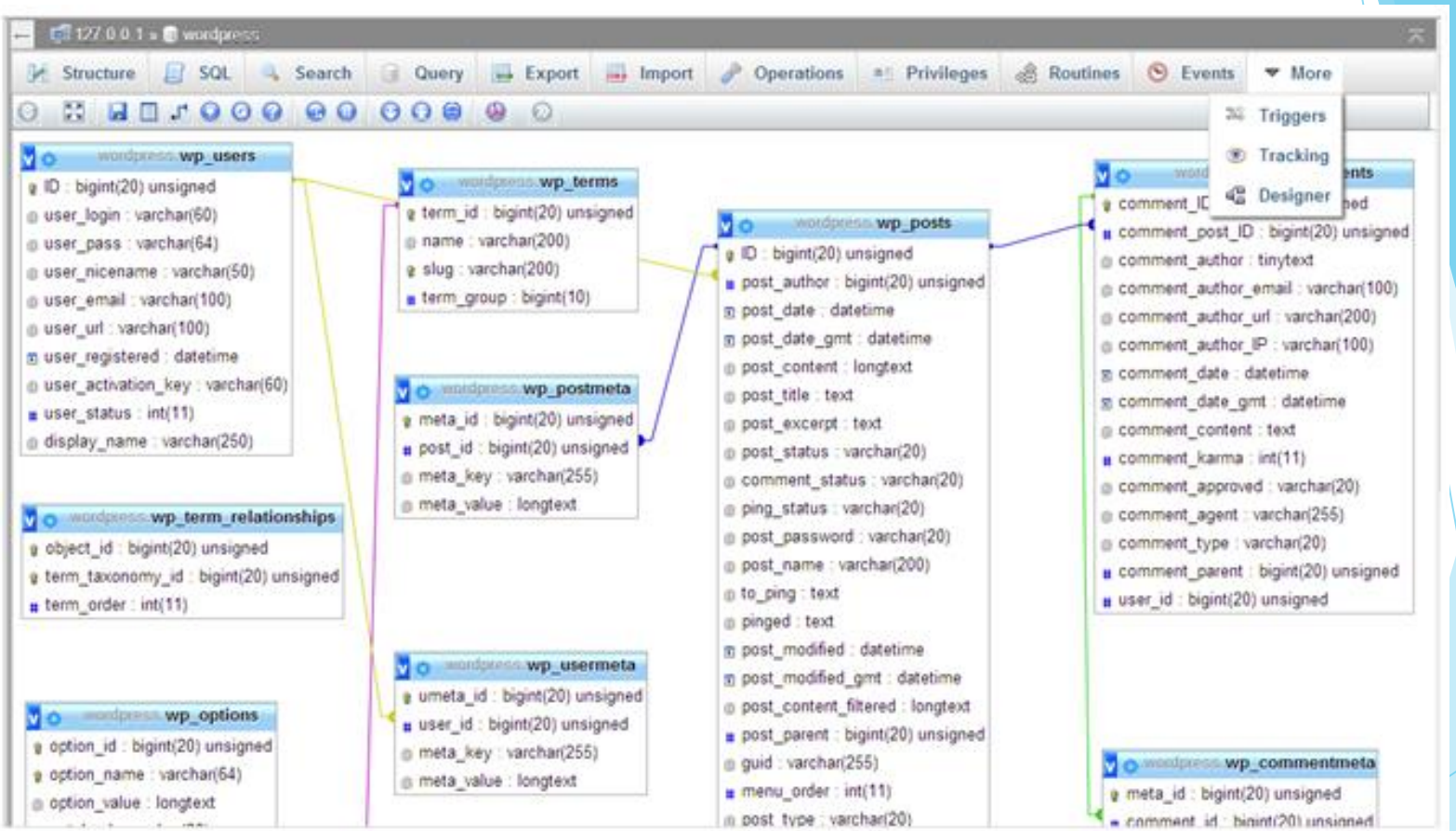
# Diagramy ERD w programach webowych





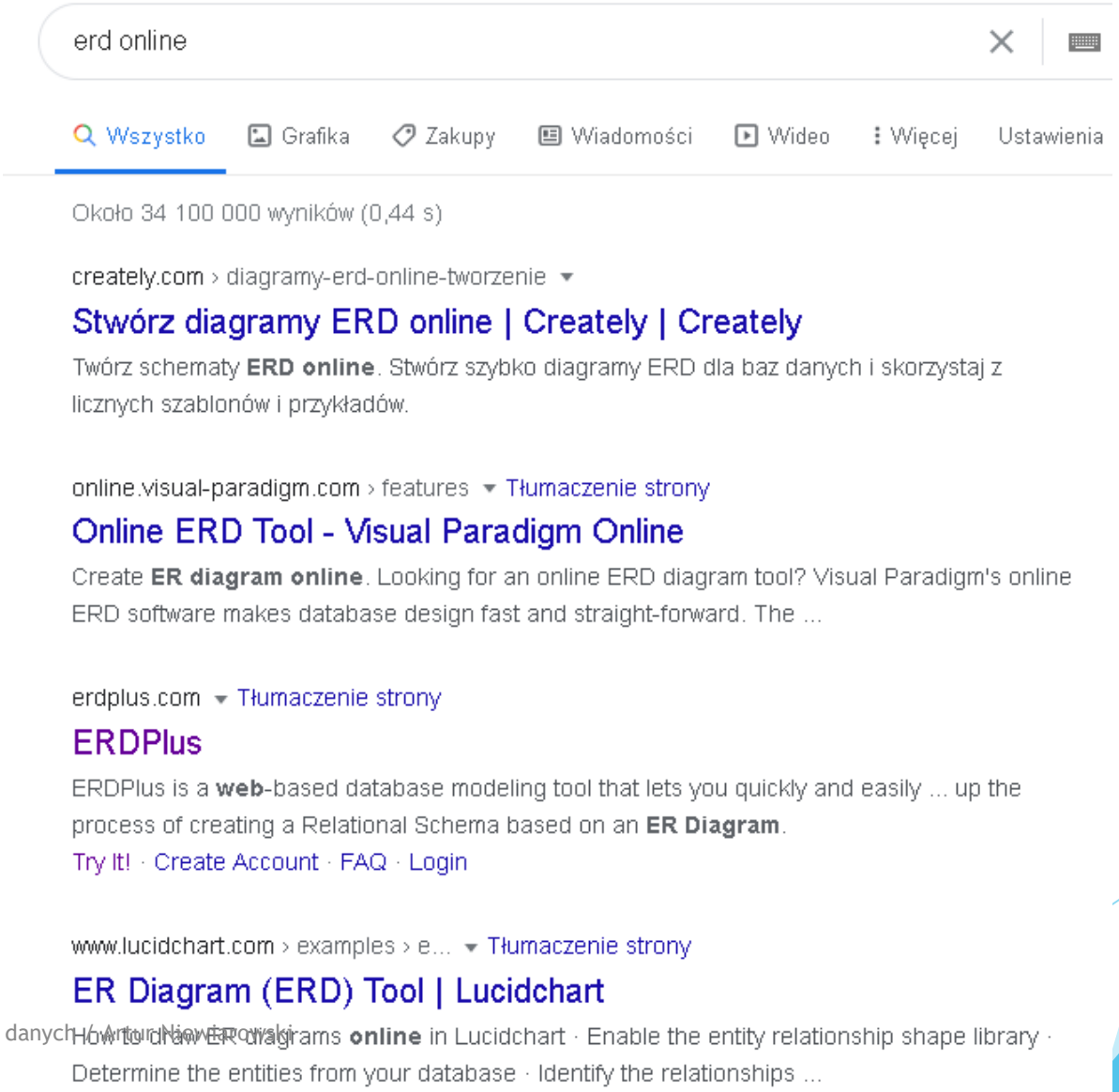
ERDPlus – diagram Chen

# Diagramy ERD w programach webowych



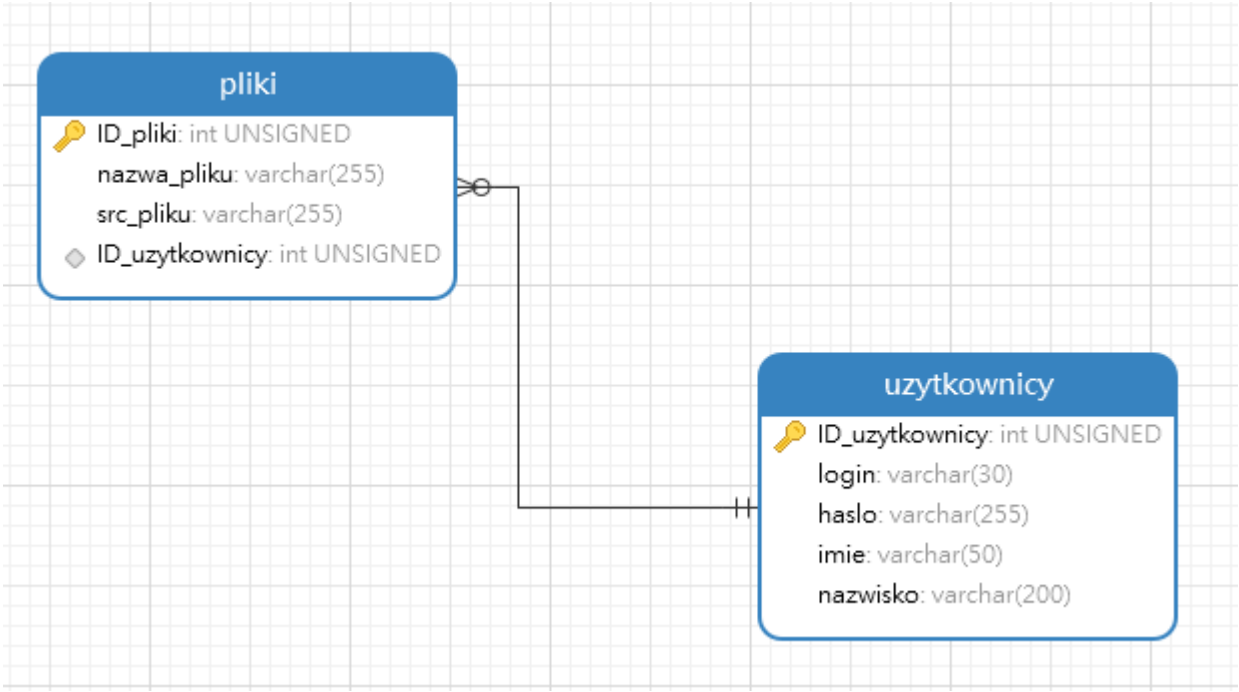

phpMyAdmin

# Diagramy ERD w programach webowych



# Przykład schematu (1)

Użytkownik posiada pliki. Jeden plik może posiadać wyłącznie jeden użytkownik. Relacja jeden-do-wielu.

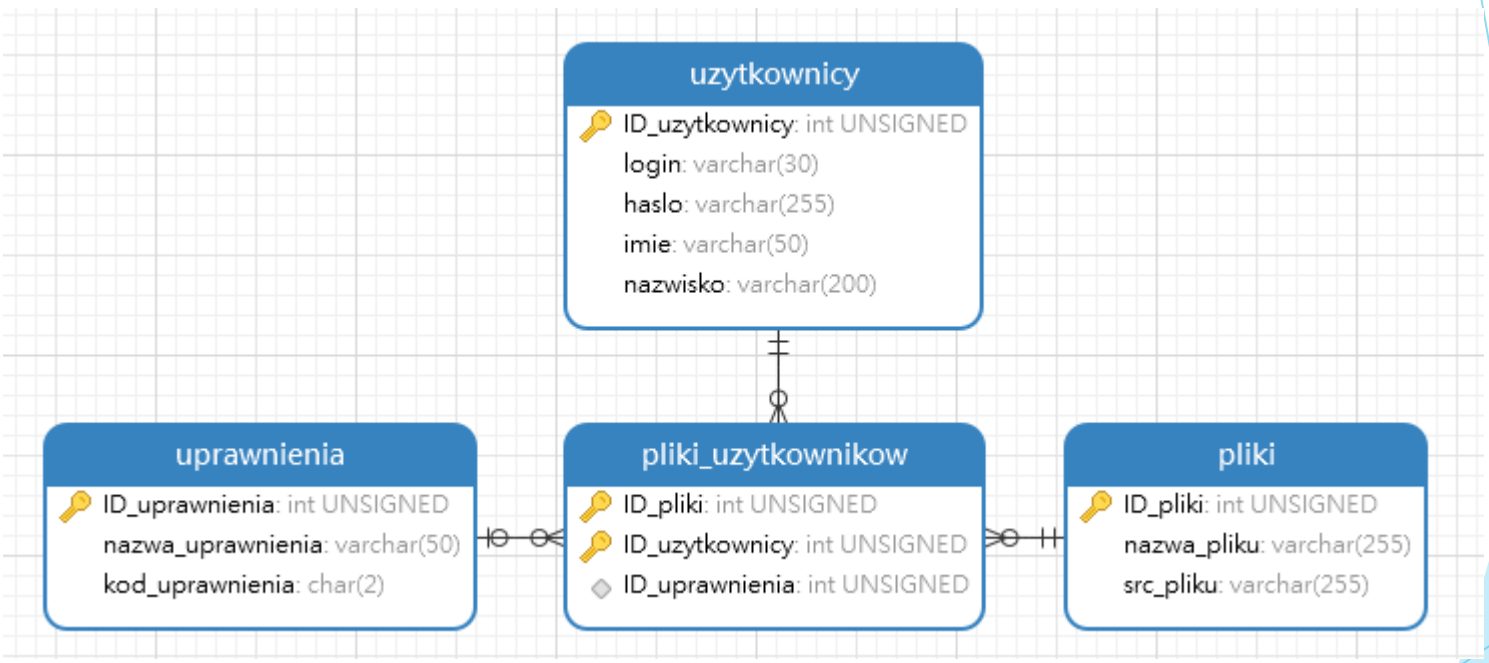


# Przykład schematu (2)

Użytkownik posiada pliki. Jeden plik może posiadać wielu użytkowników. Relacja wiele-do-wielu.

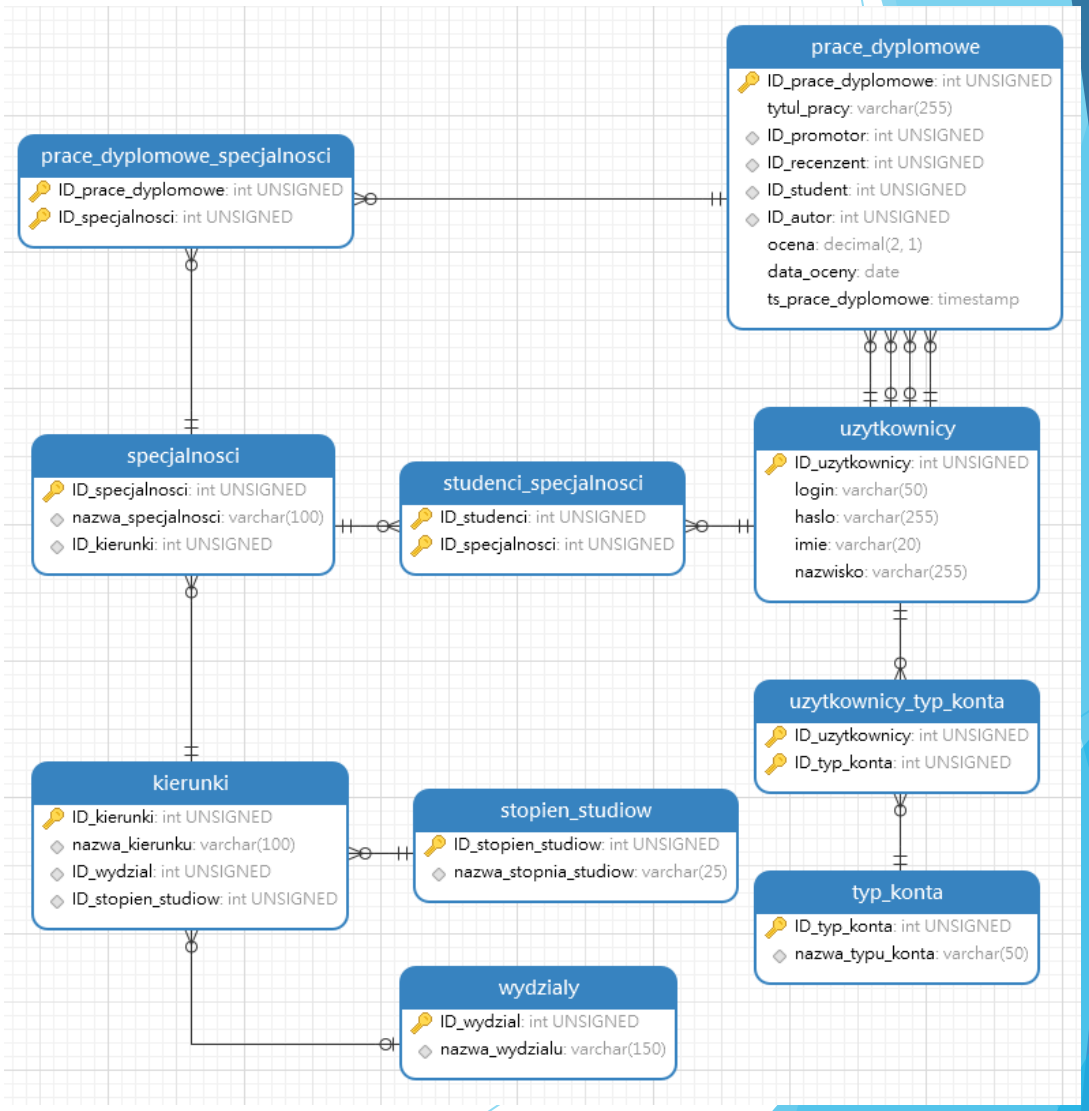
# Przykład schematu (3)

Użytkownik posiada pliki. Jeden plik może posiadać wielu użytkowników. Relacja wiele-do-wielu. Każdy użytkownik może mieć odpowiednio zdefiniowane uprawnienia do plików.



# Przykład schematu (4)

Prosty program obsługujący proces dyplomowania na uczelni



# Referencje pomiędzy kluczami PK-FK

# Referencje pomiędzy kluczami PK-FK

- Referencja między kluczem głównym a kluczem obcym jest realizowana poprzez określenie związku najczęściej między dwiema tabelami w bazie danych.
- Klucz obcy w jednej tabeli jest zdefiniowany jako odniesienie do klucza głównego w innej tabeli.
- To połączenie umożliwia tworzenie relacji między rekordami tych tabel.
- Referencje pomiędzy kluczem głównym a kluczem obcym są ważne w celu utrzymania integralności danych i zapewnienia spójności w relacyjnych bazach danych.
- Można również tworzyć referencje PK-FK w ramach tej samej tabeli.
- W ramach referencji określamy konkretne akcje.

# Referencje pomiędzy kluczami PK-FK

Schemat tworzenia klucza obcego:

* [CONSTRAINT [symbol]] **FOREIGN KEY**
[index_name] (index_col_name, ...)
REFERENCES tbl_name (index_col_name,...)
[ON DELETE **reference_option**]
[ON UPDATE **reference_option**]

**reference_option**:
RESTRICT | CASCADE | SET NULL | NO ACTION

Utworzenie referencji pomiędzy tabelami umożliwia automatyczne wygenerowanie diagramu ERD na podstawie tabel

- create table table_name (ID int, …,…, ID_fk int, *);
- alter table table_name add *;
- alter table table_name drop foreign key constr_name_fk;

# Referencje pomiędzy kluczami PK-FK

Akcje na referencjach w ramach klucza obcego umożliwiają określenie, jak baza danych powinna reagować na próbę usunięcia rekordu lub zmiany wartości komórki, do której odnosi się istniejący klucz obcy.

**reference_option**:
RESTRICT | CASCADE | SET NULL | NO ACTION

Akcje na referencji:

# Referencje pomiędzy kluczami PK-FK

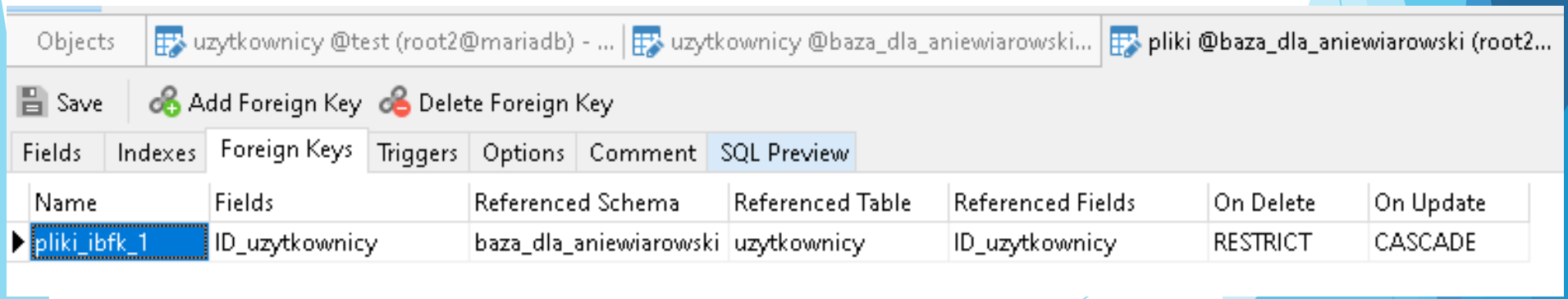
Przykład SQL DDL

```
CREATE TABLE `pliki` (
  `ID_pliki` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `ID_uzytkownicy` int(11) unsigned NOT NULL,
  `nazwa_pliku` varchar(100) NOT NULL,
  `src_pliku` varchar(250) NOT NULL,
  PRIMARY KEY (`ID_pliki`),
  KEY `fk1_idx` (`ID_uzytkownicy`),
  CONSTRAINT `pliki_ibfk_1` FOREIGN KEY (`ID_uzytkownicy`)
 REFERENCES `uzytkownicy` (`ID_uzytkownicy`)
ON UPDATE CASCADE
) ;
```

# Referencje pomiędzy kluczami PK-FK

Przykład SQL DDL oraz ustawień w interfejsie graficznym

```
CREATE TABLE `pliki` (
  `ID_pliki` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `ID_uzytkownicy` int(11) unsigned NOT NULL,
  `nazwa_pliku` varchar(100) NOT NULL,
  `src_pliku` varchar(250) NOT NULL,
  PRIMARY KEY (`ID_pliki`),
  KEY `fk1_idx` (`ID_uzytkownicy`),
  CONSTRAINT `pliki_ibfk_1` FOREIGN KEY (`ID_uzytkownicy`)
 REFERENCES `uzytkownicy` (`ID_uzytkownicy`)
ON UPDATE CASCADE
) ;
```



# Dodatkowe materiały

- https://youtu.be/MPzLYr5BbXQ