



# Bazy danych SQL

Tomasz Lis

v. 2020-07-11



# Plan zajęć

1. Podstawowe pojęcia
2. Język SQL – DDL
3. Język SQL – DML
4. Język SQL – DQL – podstawy zapytań
5. Złączenia zbiorów
6. Normalizacja
7. Złączenia tabel
8. Agregacje

## Baza danych - pojęcia

**Baza danych** – Uporządkowany zbiór danych o określonej strukturze, zapisanych na nośniku danych w sposób trwały (np. pamięci komputera).

**Model danych** – spójny zestaw pojęć, który służy do opisywania danych, związków między nimi.



# Rodzaje modelu danych

**Hierarchiczny** – dane uporządkowane w postaci drzew

**Sieciowy** – dane zorganizowane za pomocą grafów

**Obiektowy** – dane reprezentowane w postaci klas

**Logiczny (dedukcyjny)** – model danych oparty o język logiczny

**Dokument** – dane zorganizowane w postaci dokumentów np. JSON

**Klucz-Wartość** – model danych oparty o asocjacje

**Relacyjny**

**Relacyjno-objektowy**



## Relacyjny model danych - definicja

**Relacyjny model danych** jest oparty o klasyczną **Teorię mnogości**, którą znamy z matematyki jako naukę o zbiorach.

**Relacja** – fragment rzeczywistości tożsamy ze zbiorem elementów (bytów). W relacyjnym modelu danych reprezentowany w postaci tabel. Tabele składają się z wierszy i kolumn. W relacji kolumny i wiersze są unikalne. Każdy element to dana elementarna (niepodzielna semantycznie)



# Relacyjny model danych - pojęcia

**Krotka** – pojęcie tożsame z wierszem w tabeli

**Dziedzina** - zbiór wartości atomowych, które mogą przyjmować elementy atrybutu (kolumny).

**Klucz główny** – klucz składający się z jednego lub wielu elementów reprezentujących krotkę. Wartości klucza muszą być unikalne dla całej tabeli, innymi słowy nie mogą istnieć 2 krotki tej samej tabeli o tych samych kluczach głównych.



## Relacyjny pomiędzy tabelami

**Klucz obcy** – klucz składający się z jednego lub więcej elementów, o wartościach ze zbioru kluczy głównych innej tabeli. Klucz ten określa związek pomiędzy tabelami.

Tabela może relacjonować do jednej lub wielu tabel. W przypadku relacji do wielu tabel stosowane jest wiele kluczy obcych.

Relacje pomiędzy tabelami mogą zostać zrealizowane w jeden z 3 sposobów.





## Jeden do jednego (1-1)

W przypadku relacji 1-1, klucz główny tabeli A będzie referencjonowany przez dokładnie jeden klucz obcy tabeli B, natomiast klucz główny tabeli B będzie wskazywany przez dokładnie jeden klucz obcy tabeli A.

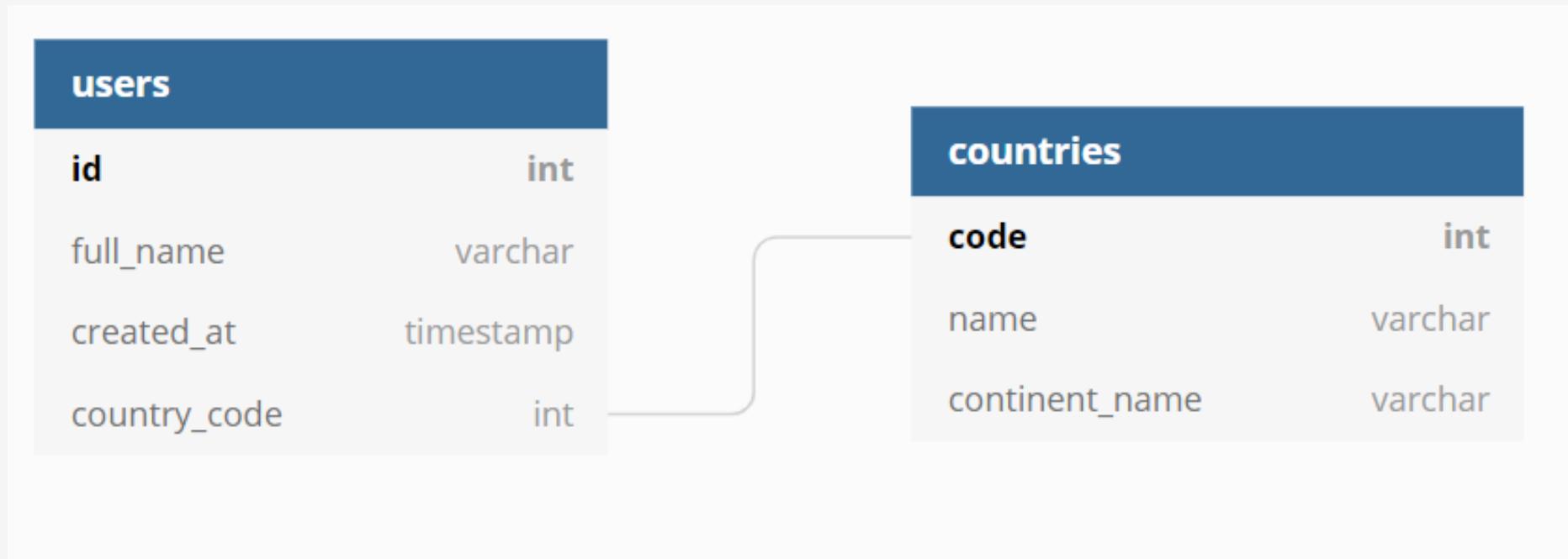
W przypadku takiego złączenia klucz główny i klucz obcy używają dokładnie tej samej kolumny.

users		address	
<b>id</b>	int	<b>id</b>	int
full_name	varchar	country_code	int
created_at	timestamp	city	varchar
		street	varchar
		street_nr	varchar



## Jeden do wielu (1-N)

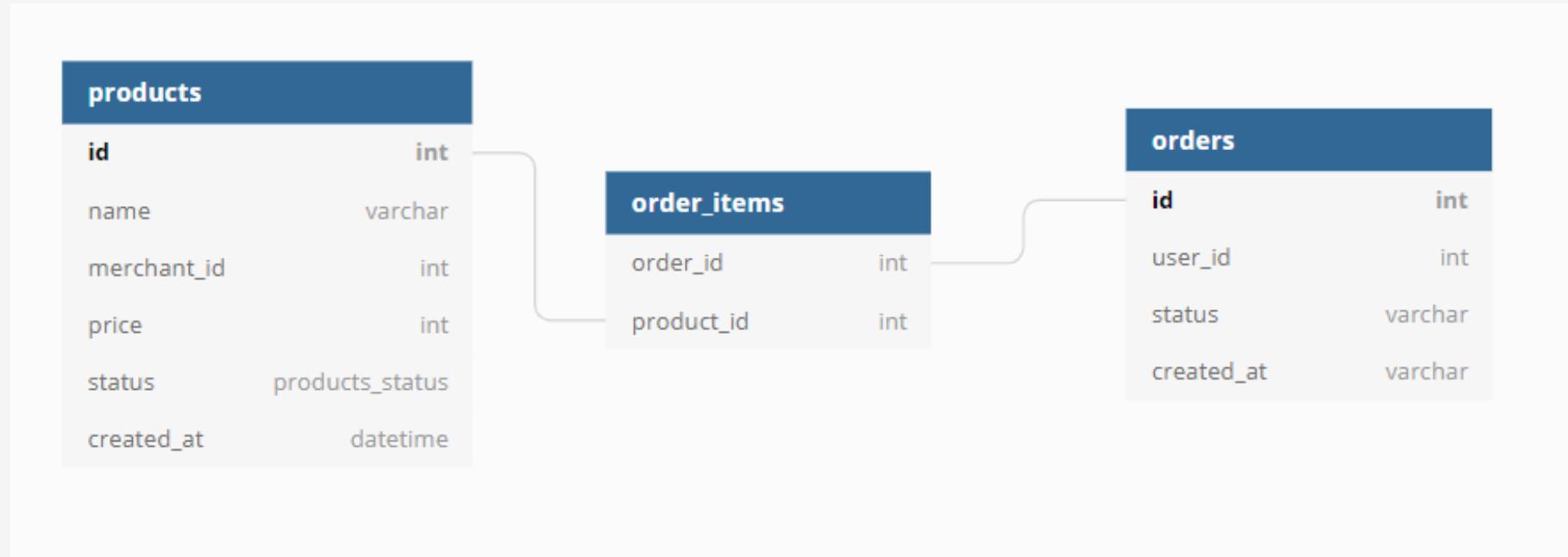
Relację jeden do wielu realizuje się zazwyczaj poprzez wskazanie klucza głównego tabeli A poprzez klucz obcy tabeli B. Tabela A natomiast nie posiada obcego do tabeli B.





## Wiele do wielu (N-N)

Najczęstszą implementacją tej relacji jest wprowadzenie dodatkowej tabeli łączącej tabelę A i B. Tabela łącząca A\_B zawiera klucze obce do tabel A i B. Para kluczy obcych często jest również kluczem głównym tabeli łączącej.





# Ćwiczenie

## Instalacja środowiska

## Zadanie A1

W ramach kursu będziemy korzystać z zadań przygotowanych w [Repozytorium](#)  
Wykonujemy zadania 1, 2 oraz 3.1 lub 3.2





# Język SQL

## Pojęcia ogólne



## Co to jest SQL

**SQL** - Structured Query Language – strukturalny język zapytań, który zapewnia komunikację między użytkownikiem lub aplikacją, a relacyjną bazą danych. Język dzielimy na:

- DCL – język kontroli bazy danych
- DDL – język definiowania schematu
- DML – język modyfikacji danych
- DQL – język zapytań do danych



## Typy danych

Typy danych dzielimy na:

- Typy Numeryczne
- Typy czasu i daty
- Typy znakowe
- Typy geometryczne (np. GIS)
- Typy nie-releacyjne (np. JSON)



## Typy numeryczne

Typy numeryczne dzielimy na:

- Typy zmiennoprzecinkowe – **FLOAT**, **DOUBLE** – float przechowuje 4 byte'y, double 2 razy więcej byte'ów
- Typy o określonej precyzyji – **DECIMAL**, **NUMERIC** – w tych typach określamy liczbę precyzyji oraz skalę np. **NUMERIC(5, 2)** oznacza pozwala przechowywać dane pomiędzy -999.99 a 999.99
- Typy całkowite – **TINYINT**, **SMALLINT**, **MEDIUMINT**, **INT**, **BIGINT** – przechowują odpowiednio 1 (**TINYINT**), 2 (**SMALLINT**), 3 (**MEDIUMINT**), 4 (**INT**), 8 (**BIGINT**) byte'ów danych
- Więcej informacji zdobędziesz na stronie dokumentacji [Numeric types](#)



## Typy daty i czasu

Typy daty i czasu dzielimy na:

- DATE – typ daty w formacie RRRR-MM-DD
- TIME – typ czasu w formacie hh:mm:ss
- DATETIME – typ daty wraz z czasem w formacie
  - RRRR-MM-DD hh:mm:ss
- TIMESTAMP – typ znacznika czasowego w formacie
  - YYYY-MM-DD hh:mm:ss[.fraction]
- YEAR – typ danych który przechowuje sam rok

Więcej informacji zdobędziesz na stronie dokumentacji [Date and Time Types](#)



## Typy Znakowe.

Typy znakowe dzielimy na:

- CHAR – ciąg znaków o zadanej długości (0-255 znaków)
- VARCHAR – ciąg znaków o zmiennej długości
- BINARY – binarny ciąg o zadanej długości (0-255)
- VARBINARY – binary ciąg o zmiennej długości
- BLOB – duży obiekt binarny
- TEXT – duży obiekt znakowy
- ENUM – typ enumeracyjny, przechowuje tylko zdefiniowane wcześniej wartości

Więcej informacji zdobędziesz na stronie dokumentacji [String types](#)



# Data Definition Language

## Tworzenie i używanie bazy danych

Aby użyć bazy danych należy ją utworzyć.

Możemy użyć do tego użytkownika głównego (root) ale w życiu codziennym każdy członek zespołu ma własne konto.



## Użyteczne komendy

**CREATE DATABASE** – utworzenie bazy danej o zadanej nazwie. Więcej informacji znajdziesz [tu](#)

**CREATE USER** – komenda pozwala na utworzenie użytkownika. Aby dowiedzieć się więcej na temat składni i dopuszczalnych parametrów sprawdź [dokumentację](#)

**GRANT** – komenda do nadawania uprawnień. Dostępne opcje można sprawdzić w [dokumentacji](#)

**USE** – przełączanie schematu



# Tworzenie tabeli

Tabele tworzymy przy pomocy komendy CREATE TABLE.

Przykład:

```
CREATE TABLE IF NOT EXISTS użytkownik (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    login VARCHAR(100),
    ostatnie_logowanie TIMESTAMP
);
```

Więcej informacji w [dokumentacji](#)



## Modyfikacja tabeli

Aby zmodyfikować wcześniej utworzoną tabelę  
należy użyć komendy ALTER TABLE, np.:

ALTER TABLE uzytkownik RENAME TO uzytkownik\_systemu;

ALTER TABLE uzytkownik\_systemu DROP COLUMN ostatnie\_logowanie;

ALTER TABLE uzytkownik\_systemu ADD INDEX loginy(login);

Więcej informacji w [dokumentacji](#).



## Usuwanie tabeli

Utworzone tabele można oczywiście również usuwać przy pomocy komendy DROP TABLE.

Przykład:

```
DROP TABLE IF EXISTS uzytkownik_systemu;
```

Więcej informacji w [dokumentacji](#).





# Ćwiczenie 2

Zakładamy pierwszą bazę danych

## Ćwiczenie 2

Wykonujemy zadania 4, 5, 6 z listy [A1](#)  
Po dokonczeniu pierwszej listy zaczynamy  
zadania z listy [B1](#)





# Data Manipulation Language

## Wstawianie krotek

Do wstawiania krotek służy komenda **INSERT INTO**.

Jej podstawowa składnia wygląda następująco:

**INSERT INTO** użytkownik **VALUES** (1, 'tomasz', now());

**INSERT INTO** użytkownik

**VALUES** (2, 'maciej', '2020-07-07 08:51:00.0000');

Więcej informacji w [dokumentacji](#).



# Usuwanie danych

Do usuwania krotek służy komenda DELETE FROM.

Jej podstawowa składnia wygląda następująco:

**DELETE FROM użytkownik WHERE id = 1;**

Instrukcja WHERE służy do ograniczenia ilości krotek, które będą obsłużone przez zapytanie. Taki wybór nazywamy selekcją.

Więcej informacji w [dokumentacji](#).



## Aktualizacja danych

Do aktualizacji danych służy komenda UPDATE. Jej podstawowa składnia wygląda następująco:

**UPDATE uzytkownik**

**SET ostatnie\_logowanie = now()**

**WHERE id = 1;**

Więcej informacji w [dokumentacji](#).



## Usunięcie wszystkich krotek

Język umożliwia nam również na skrócony sposób usunięcia krotek. Służy do tego komenda TRUNCATE. Wykorzystanie tej składni ma też aspekt wydajnościowy. Jest ona szybsza ponieważ omija wywołanie triggerów ON DELETE. Wyzwalacze (triggery) nie są elementem dzisiejszych zajęć. Warto jednak wiedzieć, że TRUNCATE TABLE jest szybsze od DELETE wszystkich wierszy. Jej podstawowa składnia wygląda następująco:

**TRUNCATE TABLE a;**



Więcej informacji w [dokumentacji](#).



# Ćwiczenie 3

## Zapełnianie bazy danych

## Ćwiczenie 3

Wykonujemy zadania z listy [B2](#)





# Data Query Language

## Selekcja i projekcja danych

Kluczowym elementem języka SQL są zapytania do wyszukiwania i prezentacji danych. Służy do tego komenda SELECT. W podstawowej formie składnia wygląda następująco:

```
SELECT * FROM uzytkownik WHERE login = 'tomasz';
```

Więcej informacji w [dokumentacji](#).



## Projekcja elementy składni

W ramach projekcji można wybrać podzbiór atrybutów. Atrybuty można również opisać nazwą dedykowaną dla projekcji przy użyciu aliasu oraz instrukcji AS:

```
SELECT DISTINCT  
    login,  
    ostatnie_logowanie AS `Ostatnie logowanie`  
FROM rejestr_logowania;
```

Jeśli chcemy otrzymać jedynie unikalne wiersze prezentowane przez zapytanie użyjemy instrukcji DISTINCT.

Więcej informacji w [dokumentacji](#).



## Określanie warunku

Aby wybrać dane, których oczekujemy w wyniku zapytania potrzebujemy operatorów. Operatory dzielimy:

- logiczne (AND, OR, NOT)
- Arytmetyczne (\*, /, +, -)
- Porównania (=, <, >, <=, >=, <>)
- SQL (BETWEEN ... AND ..., LIKE, IN, IS NULL)

Więcej informacji w dokumentacji [Functions and operators](#) oraz [Operators precedence](#).



## Sortowanie i ograniczanie wyniku zapytania

W niektórych przypadkach należy posortować wyniki zapytania. Do sortowania wyników służy instrukcja ORDER BY. Sortować można w kolejności odwrotnej (DESC). Kolejność sortowania określa się bardzo podobnie jak projekcję:

```
SELECT login, czas_logowania FROM rejestr_logowania  
ORDER BY czas_logowania DESC, login LIMIT 5;
```

Można ograniczyć wyniki zapytania przy pomocy instrukcji LIMIT.



# LIKE

Operator **LIKE** jest używany w klauzuli **WHERE** do wyszukiwania określonego wzoru w kolumnie.

W połączeniu z operatorem LIKE istnieją dwa **symbole wieloznaczne**:

% - znak procentowy oznacza zero, jeden lub wiele znaków

\_ - podkreślenie oznacza pojedynczy znak





# Ćwiczenie 4

## Selekcja i projekcja danych

## Ćwiczenie 3

Wykonujemy zadania z listy [B3](#)





# Operacje na zbiorach

# Operacje na zbiorach

Podobnie jak teoria mnogości, tak i standard SQL opisuje operacje na zbiorach:

- UNION (oraz UNION ALL) – suma zbiorów, zwana też łączeniem zbiorów.
- INTERSECT – część wspólna zbiorów, zwana również iloczynem zbiorów
- EXCEPT – różnica zbiorów

MySQL nie wspiera składni INTERSECT ani EXCEPT. Można posłużyć się natomiast instrukcjami EXISTS, NOT EXISTS. Konstrukcje te są bardziej złożone niż zwykły SELECT dlatego omówimy je na osobnych slajdach.



# Operacje na zbiorach

Dla ułatwienia rozważać będziemy dwa zbiory a(1,2,3) oraz b (2,3,4):

```
CREATE TABLE a  
    id TINYINT PRIMARY KEY  
);  
INSERT INTO a VALUES (1), (2), (3);  
CREATE TABLE b  
    id TINYINT PRIMARY KEY  
);  
INSERT INTO b VALUES (2), (3), (4);
```



# UNION i UNION ALL

```
SELECT * FROM a  
UNION  
SELECT * FROM b;
```

	<b>id</b>
	1
	2
	3
▶	4

```
SELECT * FROM a  
UNION ALL  
SELECT * FROM b;
```

	<b>id</b>
▶	1
	2
	3
	2
	3
	4



# INTERSECT

```
SELECT * FROM a WHERE  
EXISTS (SELECT 1 FROM b WHERE a.id = b.id);
```

	<code>id</code>	...
▶	2	
	3	



# EXCEPT

```
SELECT * FROM a WHERE  
NOT EXISTS (SELECT 1 FROM b WHERE a.id = b.id);
```

	<b>id</b>
▶	1





# Ćwiczenie 5

## Operacje na zbiorach

## Ćwiczenie 5

Wykonujemy zadania z listy [B4](#)





# Normalizacja



# Normalizacja

W relacyjnych bazach danych unika się redundancji przechowywanych informacji. System w którym istnieje jedna tabela z jedną kolumną jest antywzorcem. Rozdziela się np. użytkowników od zbioru numerów telefonu, adresu, itp. na osobne kolumny i tabele. Proces ten nazywamy **dekompozycją**, kluczowym procesem normalizacji. **Normalizacja** natomiast jest to proces sprowadzania do jednej z postaci normalnych. Omówimy je na kolejnych slajdach.



## Pierwsza postać normalna

**Pierwsza postać normalna** opisuje sytuację w której każdy atrybut jest skalarem niepodzielnym na prostsze elementy.

Dla przykładu postać normalną spełnia schemat:

PRACOWNIK( **id\_pracownik** , imię, nazwisko, miejscowość, ulica, nr domu, dział, zawód, wynagrodzenie)



## Druga postać normalna

**Druga postać normalna** rozbudowuje **pierwszą postać normalną** o następujące wymaganie:

Żadna kolumna poza kluczem głównym nie może zależeć od żadnego innego potencjalnego klucza.

Aby sprowadzić schemat z poprzedniego slajdu do drugiej postaci normalnej należy utworzyć następujące tabele:

PRACOWNIK(**id\_pracownik**, imię, nazwisko, zawód, wynagrodzenie, **id\_dział**)

ADRES(**id\_pracownik**, miejscowość, ulica, numer\_domu)

DZIAŁ(**id\_dział**, nazwa)



## Trzecia postać normalna

Schemat który spełnia **trzecią postać normalną** musi spełniać wszystkie warunki **Pierwszej i Drugiej postaci normalnej** oraz wszystkie atrybuty w tabeli są niezależne od innych.

Zakładając, że wiele pracowników mieszka w jednej miejscowości:

PRACOWNIK(**id\_pracownik**, imię, nazwisko, zawód, wynagrodzenie, **id\_dział**)

ADRES(**id\_pracownik**, **id\_miejscowość**, ulica, numer\_domu)

DZIAŁ(**id\_dział**, nazwa)

MIEJSCOWOŚĆ(**id\_miejscowość**, nazwa)



## Postać normalna Boyce'a-Codda

Jest to postać normalna, która wymaga aby każdy atrybut wewnątrz relacji zależał tylko od jednego klucza relacji. W tej postaci normalnej zakłada się istnienie więcej niż jednego klucza podstawowego. Każdy klucz może zawierać więcej niż jeden atrybut.

W uproszczeniu jest to schemat, który wprowadza bardzo dużą ilość słowników;

PRACOWNIK(**id\_pracownik**, id\_imię, nazwisko, id\_zawód, wynagrodzenie, id\_dział)

ADRES(**id\_pracownik**, id\_miejscowość, id\_ulica, numer\_domu)

DZIAŁ(**id\_dział**, nazwa)

MIEJSCOWOŚĆ(**id\_miejscowość**, nazwa)

ZAWÓD(**id\_zawód**, nazwa)

IMIĘ(**id\_imię**, nazwa)

ULICA(**id\_ulica**, nazwa)



# Zakładanie klucza obcego

Przy normalizacji schematu bardzo powszechną praktyką będzie zakładanie klucza obcego pomiędzy tabelami. Jeśli tabela już istnieje można wykonać to komendą alter:

```
ALTER TABLE tabela1 ADD CONSTRAINT tabela1_tabela2_fk FOREIGN KEY (id_tabela2) REFERENCES tabela2(id);
```

Jeśli tworzymy nową tabelę można to zrobić jako definicję jednej z kolumn:

```
CREATE TABLE tabela4(
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    id_tabela2 BIGINT,
    FOREIGN KEY (id_tabela2) REFERENCES do_usuniecia2(id)
);
```

Lub w skróconej formie:

```
CREATE TABLE tabela3(
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    id_tabela2 BIGINT REFERENCES do_usuniecia2(id)
);
```



## Pomocna konstrukcja INSERT

W życiu codziennym rzadko udaje się od razu zaprojektować odpowiedni schemat danych. Gdy zapadnie decyzja o dekompozycji tabeli, która istnieje w systemie produkcyjnym należy przenieść dane do innych tabel w sposób zautomatyzowany.

Pomoże nam w tym następująca konstrukcja:

INSERT INTO adres

```
SELECT id, miejscowosc, ulica, numer_domu FROM pracownik;
```





# Ćwiczenie 6

## Normalizacja i dekompozycja

## Ćwiczenie 6

Wykonujemy zadania z listy [C1](#)





# Złączenia tabel



## Rodzaje złączeń

Połączenia pomiędzy tabelami dzielimy na:

Wewnętrzne (INNER JOIN) – wynikiem złączenia będą tylko te rekordy, które spełniają warunek złączenia.

Prawostronne (RIGHT JOIN) – wynikiem złączenia będą rekordy, które spełniają warunek złączenia, oraz reszta rekordów tabeli po prawej stronie

Lewostronne (LEFT JOIN) – wynikiem złączenia będą wszystkie rekordy tabeli po lewej stronie oraz rekordy tabeli po prawej stronie, które spełniają warunek złączenia.

Iloczyn kartezjański (CROSS JOIN) – Wynikiem będą wszystkie rekordy obu tabel we wszystkich możliwych kombinacjach

Pomocny link: [sqlpedia](#)

# Składnia zapytania

Złączenie tabel przy pomocy zapytania SELECT wygląda następująco:

```
SELECT imie, nazwisko, a.miejscowosc  
FROM  
    pracownik p  
LEFT JOIN  
    adres a  
USING(id);
```

```
SELECT imie, nazwisko, a.miejscowosc  
FROM  
    pracownik p  
LEFT JOIN  
    adres a  
ON a.id_pracownik = p.id;
```

Obje składnie są analogiczne, przykład po lewej stronie wykorzystuje charakterystykę relacji 1-1.

Więcej informacji w [dokumentacji](#)





# Ćwiczenie 7

## Złączenia tabel

## Ćwiczenie 7

Wykonujemy zadania z listy [C2](#)





# Agregacja danych

## Grupowanie danych

Składnia zapytania grupującego dane wygląda bardzo podobnie jak składnia sortowania wyników. W tym przypadku stosujemy klauzulę GROUP BY. W przypadku zapytania które grupuje i sortuje wyniki, grupowanie musi występować przed sortowaniem.

Przykład zapytania:

```
SELECT a.miejscowosc, COUNT(*) FROM pracownik p LEFT JOIN adres a ON a.id = p.id  
GROUP BY 1  
ORDER BY 2;
```

Pomocna w składni jak zwykle [dokumentacja](#)



## Agregacja wyników

W wyniku grupowania danych przy użyciu danego atrybutu możemy agregować wartości pozostałych atrybutów. Pomagają nam w tym funkcje agregujące.

Kilka przykładów funkcji:

SELECT miejscowosc,

avg(wiek), – średnia

max(wiek), – wartość maksymalna

min(wiek), – wartość minimalna

count(distinct imie), – ilość unikalnych atrybutów

group\_concat(imie) – grupowanie atrybutów po przecinku

FROM pracownik GROUP BY 1;





# Ćwiczenie 8

## Złączenia tabel

## Ćwiczenie 8

Wykonujemy zadania z listy [C3](#)





# Transakcje



# Transakcje

Sekwencja (uporządkowany zbiór) logicznie powiązanych operacji na bazie danych, która przeprowadza bazę danych z jednego stanu spójnego w inny stan spójny.



# Transakcje

SAVEPOINT

Rozpoczęcie

BEGIN

Wykonanie

COMMIT,  
ROLLBACK

