# INFORMATYKA II

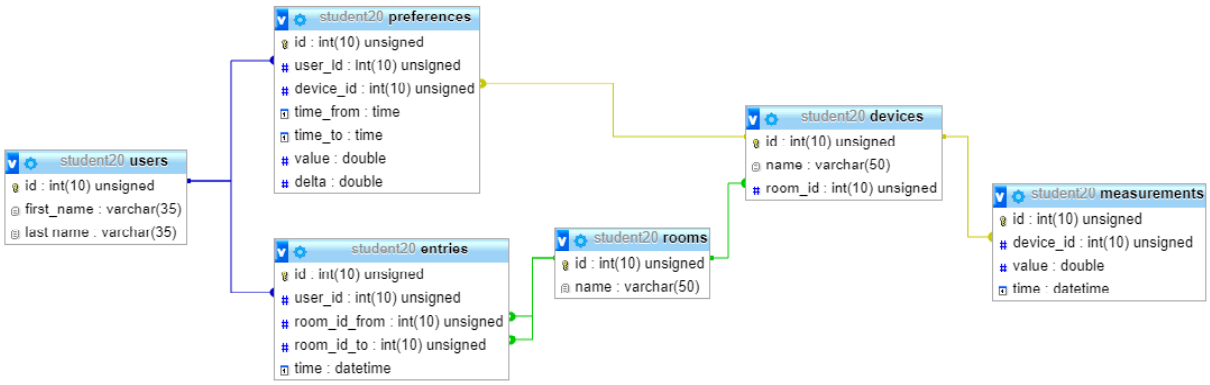
ZŁĄCZENIA W JĘZYKU SQL

## CEL LABORATORIUM

Celem zajęć jest zapoznanie się z ideą złączeń i zapytań zagnieżdżonych w języku SQL. W trakcie zajęć przedstawione zostaną różne rodzaje złączeń i zapytań zagnieżdżonych, wraz z praktycznymi przykładami. Zaprezentowane również zostaną różnice pomiędzy tymi rozwiązaniami.

## MATERIAŁY POMOCNICZE

Diagram bazy danych przedstawia się następująco:




Kod SQL generujący wykorzystywaną w trakcie zajęć bazę danych można pobrać pod adresem http://www.stawarz.edu.pl/informatyka2/smart_house.sql.

Baza danych może zostać uruchomiona na każdym komputerze, który jest zaopatrzony w system zarządzania bazami danych MySQL. Darmowym programem pozwalającym pracować z bazą danych w zaciszu domowym jest program XAMPP, który można pobrać pod adresem https://www.apachefriends.org/pl/index.html.

Program opracowany na poprzednich zajęciach (w wersji okrojonej), można pobrać pod adresem http://www.stawarz.edu.pl/informatyka2/Lab2.rar (lub w wersji ZIP)

## KLUCZE OBCE

Na poprzednich zajęciach wprowadzono pojęcie klucza głównego. Dla przypomnienia – kluczem głównym nazywamy kolumnę (lub kolumny), które mogą zawierać wyłącznie unikalne wartości i służą do identyfikacji rekordu w tabeli.

Na diagramie bazy danych można zaobserwować również połączenia. Zwróć uwagę, że łączą one klucz główny jednej tabeli (nazwijmy ją *tabelą A*) z kolumną innej tabeli (*tabela*

*B*). Takie połączenie oznacza, że wartości zapisane w kolumnie tabeli B bezpośrednio odnoszą się do wartości klucza głównego tabeli A. Kolumnę tabeli B nazywamy **kluczem obcym**.

Przykładowo, pole *id* tabeli *users* jest kluczem głównym tej tabeli. Pole *user_id* tabeli *entries* jest natomiast kluczem obcym tabeli *entries*. Tabela może mieć więcej niż jeden klucz obcy. Dla przykładu, w tabeli *entries*, kluczami obcymi są kolumny *user_id*, *room_id_from* oraz *room_id_to*. Zwróć uwagę, że klucze obce *room_id_from* oraz *room_id_to* odwołują się do tej samej kolumny – klucza głównego *rooms*, czyli kolumny *id*.

## ZŁĄCZENIA

Klucze obce pozwalają tworzyć **relacje** pomiędzy tabelami. Aby uwzględnić relację w zapytaniu języka SQL, należy zastosować mechanizm zwany **złączeniem**. Złączenie tworzy się poprzez użycie klauzuli **JOIN**. Rozszerzona o tę klauzulę składnia polecenia **SELECT** wygląda następująco:

```
SELECT kolumna1[, kolumna2[, kolumnaN] ]
FROM tabela1
JOIN tabela2
ON kluczObcy = kolumna
[ WHERE warunek ]
[ ORDER BY kolumna[ ASC | DESC ] ];
```

Rozważmy przykładowy problem. Załóżmy, że chcemy **pobrać z bazy danych informacje o urządzeniach znajdujących się w przedpokoju**. Skonstruujmy nasze zapytanie krok po kroku. Rozpocznijmy od prostego zapytania, które zwraca wszystkie informacje o wszystkich urządzeniach:

```
SELECT * FROM devices;
```

Następnie rozbudujmy to zapytanie, łącząc tabelę *devices* z tabelą *rooms*. Jak możemy zauważyć na diagramie, połączone są kolumny *room_id* tabeli *devices* oraz kolumna *id* tabeli *rooms*. Uwzględnijmy ten fakt w naszym zapytaniu:

```
SELECT * FROM devices
JOIN rooms
ON room_id = rooms.id;
```

Zwróć uwagę, że przed kolumną *id* znajduje się nazwa tabeli (*rooms*), oddzielona od *id* kropką. Jest tak dlatego, że zarówno tabela *devices*, jak i tabela *rooms*, zawierają kolumnę *id*. Możesz spróbować wywołać to zapytanie w swojej aplikacji i przeanalizować wynik.

Chcąc ograniczyć się wyłącznie do urządzeń znajdujących się w przedpokoju, do zapytania po prostu dodalibyśmy klauzulę *WHERE*. Ostateczna postać zapytania wyglądałaby więc następująco:

```
SELECT * FROM devices
JOIN rooms
ON room_id = rooms.id
WHERE rooms.name = „Przedpokój”;
```

Ponownie zwróć uwagę, że kolumna *name* jest poprzedzona oznaczeniem tabeli (*rooms*). Może się zdarzyć, że po wypróbowaniu tego zapytania, nie uzyskasz żadnych wyników pomimo tego, że zapytanie będzie poprawne. W takim wypadku upewnij się, że poprawnie napisałaś/napisałeś „*Przedpokój*”. Jeżeli nie wiesz jak nazwa ta jest zapisana w bazie danych, zawsze możesz uzyskać odpowiedź zapytaniem:

```
SELECT name FROM rooms;
```

## RODZAJE ZŁĄCZEŃ

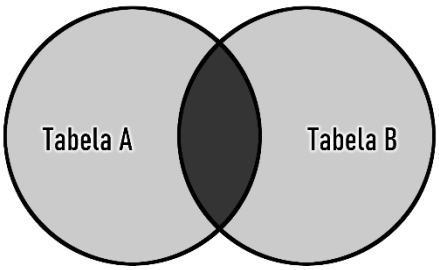
Istnieją cztery rodzaje złączeń. Przy pierwszym czytaniu tej instrukcji możesz ominąć ten rozdział. Wrócisz tu w odpowiednim czasie. Być może stanie się to podczas realizacji zadań. W tej chwili warto jedynie wiedzieć, że **domyślnie**, jeżeli nie zaznaczymy o jaki typ złączenia chodzi, SQL zastosuje **inner join**.

| NAZWA | OPIS DZIAŁANIA |
|---|---|
| **INNER JOIN** | Wybiera tylko te wiersze z tabeli pierwszej, które są złączone z wierszami w tabeli drugiej. Jeżeli wartość klucza obcego żadnego z wierszy tabeli drugiej nie wskazuje na dany wiersz tabeli pierwszej, wiersz ten nie pojawi się w wyniku. |
| **LEFT JOIN** | Wybiera wszystkie wiersze z tabeli pierwszej, złączając je z wierszami w tabeli drugiej. Jeżeli wartość klucza obcego żadnego z wierszy tabeli drugiej nie wskazuje na dany wiersz tabeli pierwszej, brakujące wartości będą zastąpione wartością *NULL*. |
| **RIGHT JOIN** | Wybiera wszystkie wiersze z tabeli pierwszej, złączając je z wierszami w tabeli drugiej (jak inner join) oraz wszystkie wiersze tabeli drugiej, których wartość klucza obcego nie wskazuje na żaden wiersz w tabeli pierwszej. |
| **FULL JOIN** | Wybiera wszystkie wiersze tabeli pierwszej i wszystkie wiersze tabeli drugiej. Jeżeli jakieś dane nie mają odpowiednika w jednej lub w drugiej tabeli, podstawiane są wartości *NULL*. |

Można zaprezentować działanie różnych rodzajów złączeń w formie rachunku zbiorów[1]. W takim wypadku tabele reprezentuje się jako zbiory, które powinny być Ci znane z lekcji matematyki. Lewy zbiór to tabela pierwsza (czyli ta posiadająca klucz obcy), prawy zbiór to tabela druga. Obszar ciemnoszary oznacza dane, które znajdą się w wyniku.

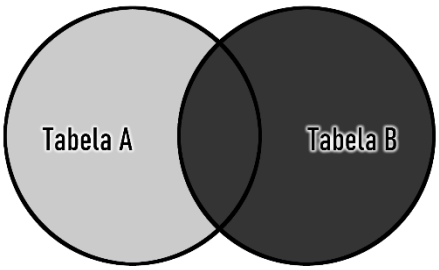
[1] W końcu baza danych to tylko zbiór danych zapisany w pewnej sformalizowanej postaci

**INNER JOIN**



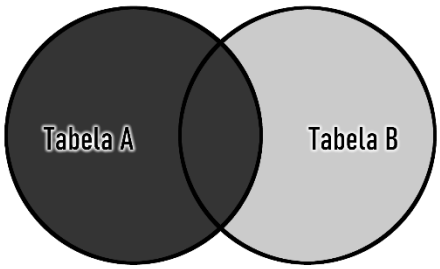

$C = (A \cap B)$

**RIGHT JOIN**



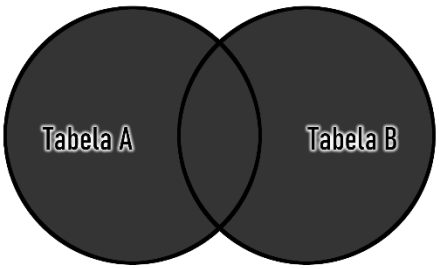

$C = (A \cap B) \cup B$

**LEFT JOIN**




$C = (A \cap B) \cup A$

**FULL JOIN**




$C = (A \cap B) \cup (A - B) \cup (B - A)$

## MODUŁ INFORMACJI O POMIARACH

Uruchom aplikację opracowaną na poprzednich zajęciach. Jeżeli jest niepełna, możesz skorzystać z okrojonej wersji, do której odnośnik znajduje się w jednym z poprzednich rozdziałów niniejszej instrukcji.

Utwórz nowy plik. Nazwij go „modułPomiarów.py". Znajdzie się w nim moduł zwracający informacje o pomiarach wykonanych przez urządzenia w domu. Przepisz do tego pliku następujący kod języka Python:

```
from moduł import *
import mysql.connector

class ModułPomiarów(Moduł):

    def __init__(self):
        self.identyfikator = "P"
        self.nazwa = "Moduł pomiarów"

    def przejmijKontrolę(self):
        print("\nModuł pomiarów:")
        print("======================")
        kursor = self.aplikacja.połączenie.cursor(dictionary=True)
        while (True):
            print("[1] Wypisz średnią temperaturę")
            print("[KONIEC] Wyjdź z modułu")

            opcja = input("Wybierz opcję z menu: ")
```

```
            if opcja.upper() == "KONIEC":
                break
            elif(opcja == "1"):
                self.średniaTemperatura(kursor)
            else:
                print("Nie ma takiej opcji w menu. Spróbuj ponownie!")
            print("\n")
        kursor.close()

    def średniaTemperatura(self, kursor):
        try:
            # Tu należy zadać odpowiednie zapytanie...

        except mysql.connector.Error as błąd:
            print("Błąd w zapytaniu SQL:",błąd)

        else:
            for wiersz in kursor:
                print(wiersz)
```

Pamiętaj, że moduł należy zarejestrować w aplikacji, zanim będziemy mogli go wykorzystać. Jeżeli nie wiesz jak to zrobić, wróć do poprzednich instrukcji.

## ŚREDNIA TEMPERATURA W DOMU

Dysponując wiedzą zdobytą na poprzednich zajęciach oraz wiedzą o złączeniach w języku SQL, jesteśmy w stanie skonstruować zapytanie zwracające nam informacje o średniej temperaturze w całym domu.

Zacznijmy od prostego zapytania, które zwróci nam wartość wszystkich pomiarów. Informacje o pomiarach przechowywane są w tabeli *measurements*, a wartość tychże, reprezentują wartości zapisane w kolumnę *value*:

```
SELECT value FROM measurements;
```

W następnym kroku ograniczmy się tylko do wartości średniej. W tym celu skorzystamy z funkcji *AVG*:

```
SELECT AVG(value) FROM measurements;
```

W jaki sposób ograniczyć się wyłącznie do pomiarów temperatury? Cóż – pomiary temperatury to te pomiary, które zostały wykonane przez... **termometr**. Nie możemy jednak zadać zapytania w następujący sposób:

```
SELECT AVG(value) FROM measurements
WHERE name = „Termometr”;
```

Dlaczego? Ponieważ w tabeli *measurements* nie ma kolumny *name*. Znajduje się ona w tabeli *devices*. W związku z tym tabele należy połączyć za pomocą klauzuli **JOIN**. Połączone ze sobą są kolumna *device_id* tabeli *measurements* oraz kolumna *id* tabeli *devices*:

```
SELECT AVG(value) FROM measurements
JOIN devices
ON device_id = devices.id
WHERE name = „Termometr”;
```

Musimy oczywiście umieścić to zapytanie w kodzie naszej aplikacji, w odpowiednim pliku i miejscu:

```
kursor.execute("SELECT AVG(value) FROM measurements "
               "JOIN devices "
               "ON device_id = devices.id "
               "WHERE name = \"Termometr\"")
```

Zwróć uwagę, że cudzysłowy należy poprzedzić ukośnikami. W przeciwnym razie język Python potraktuje je jako zakończenie i rozpoczęcie ciągu znaków i zgłosi problem ze składnią.

## ZŁĄCZENIA WIELOKROTNE

Złączenia można dokonywać pomiędzy więcej niż dwoma tabelami. W takim wypadku musimy zastosować klauzulę JOIN więcej razy.

Przykładowo, gdybyśmy chcieli ograniczyć się do średniej temperatury w salonie, musielibyśmy odwołać się także do tabeli rooms i pola name, które przechowuje nazwę pokoju. W takim wypadku nasze zapytanie miałoby następującą postać:

```
SELECT AVG(value) FROM measurements
JOIN devices
ON device_id = devices.id
JOIN rooms
ON room_id = rooms.id
WHERE devices.name = „Termometr”
AND rooms.name = „Salon”;
```

## ZAPYTANIA ZAGNIEŻDŻONE

Rozważmy następujący problem – chcemy pobrać imię i nazwisko osoby, która jako ostatnia przeszła do pomieszczenia o identyfikatorze 1. Wyjdźmy od prostego zapytania:

```
SELECT first_name, `last name` FROM users;
```

Dołączmy dane o przejściach pomiędzy pomieszczeniami. Informacje te są zapisane w tabeli *entries*. Łączenie odbywa się pomiędzy kolumną *id* tabeli *users*, a kolumną *user_id* tabeli *entries*:

```
SELECT first_name, `last name` FROM users
JOIN entries
ON user_id = users.id;
```

Teraz pozostaje ograniczyć dane do tych, które dotyczą wyłącznie mieszkańca, który jako ostatni wszedł do pokoju o identyfikatorze 1. Kolumna *entries* zawiera czas zdarzenia. Wystarczy więc z tabeli *entries* wybrać ten rekord, który ma maksymalną wartość kolumny *time* i wartość 1 w kolumnę *room_id_to*. Realizuje to poniższe zapytanie:

```
SELECT MAX(time) FROM entries
WHERE room_id_to = 1;
```

Nadal pozostaje kwestia wykorzystania tej informacji w naszym poprzednim zapytaniu. Na szczęście język SQL nam na to pozwala. Należy tylko pamiętać, żeby **podzapytanie** otoczyć nawiasami:

```
SELECT first_name, `last name` FROM users
JOIN entries
ON user_id = users.id
WHERE time = (
   SELECT MAX(time) FROM entries
   WHERE room_id_to = 1
);
```

W większości wypadków można uniknąć tworzenia zapytań zagnieżdżonych. **Unikanie** tego typu zapytań może być dobrym pomysłem, gdyż bywa, że wykonują się one znacznie **wolniej**. W razie niepewności – warto skonsultować się z dokumentacją silnika bazy danych.

Ważne jest również to, by wspomnieć o istnieniu operatora **IN**. W powyższym wypadku istnieje tylko **jeden** wiersz, który spełnia podane w zapytaniu zagnieżdżonym warunki. Co jednak, gdyby wierszy było więcej? Wtedy użycie operatora porównania („=") byłoby błędem. Zastąpić go należy słowem kluczowym *IN*. Dla bezpieczeństwa – warto to robić **zawsze**.

## ZADANIA DO SAMODZIELNEGO WYKONANIA

# ZADANIE 1

Zmień działanie funkcji *wypiszWPomieszczeniu* w taki sposób, aby użytkownik aplikacji mógł podać nazwę pomieszczenia zamiast jego identyfikatora. Na przykład, po podaniu na wejście „*Przedpokój*", program powinien wypisać dane urządzeń o identyfikatorach 6 i 7 (odpowiednio *Żyrandol* i *Termometr*). Rozwiąż problem za pomocą **jednego** zapytania do bazy danych.

# ZADANIE 2 Ⓟ

Zmień działanie funkcji *średniaTemperatura* modułu informacji o pomiarach w taki sposób, aby oprócz pokazywania średniej temperatury w całym domu, pokazywała również średnią temperaturę w każdym pomieszczeniu oddzielnie.

# ZADANIE 3

Dodaj do modułu informacji o pomiarach opcję „Wyświetl aktualną temperaturę". Po wybraniu tej opcji, użytkownikowi powinna się wyświetlić informacja o ostatniej zmierzonej temperaturze w każdym z pomieszczeń domu.

# ZADANIE 4 Ⓟ

Dodaj do modułu informacji o pomiarach opcję „Wyświetl historię pomiarów". Po wybraniu tej opcji, użytkownikowi powinna się wyświetlić lista urządzeń dostępnych w domu w formie par (identyfikator, nazwa). Następnie aplikacja powinna poprosić użytkownika o podanie identyfikatora urządzenia. Gdy użytkownik poda poprawny identyfikator, aplikacja powinna wyświetlić wszystkie pomiary wykonane przez urządzenie, chronologicznie, począwszy od najnowszego.

# ZADANIE 5 Ⓟ

Dodaj do modułu informacji o mieszkańcach opcję „Znajdź w domu". Po wybraniu tej opcji, użytkownikowi powinna się wyświetlić lista mieszkańców wraz z nazwami pokojów, w których aktualnie się znajdują. Każdy wpis powinien być w następującej postaci: „Imię nazwisko – nazwa pokoju" (np. „Adam Kowalski – Salon").

# ZADANIE 6 Ⓟ (NAGRADZANE 2 PLUSAMI)

Dodaj do modułu informacji o urządzeniach opcję „Pomiary urządzenia". Po wybraniu tej opcji, użytkownikowi powinna się wyświetlić lista urządzeń wraz z identyfikatorami. Następnie aplikacja powinna poprosić użytkownika o podanie identyfikatora. Po podaniu identyfikatora, powinny zostać wypisane wszystkie informacje o danym urządzeniu w następującej formie:

```
[identyfikator] Nazwa urządzenia w nazwa pokoju:
- wykonanych pomiarów: X
- data ostatniego pomiaru: Y
- najwyższa zmierzona wartość: W
- najniższa zmierzona wartość: Z
```


| | |
|---|---|
| Autor: | Mgr inż. Paweł Stawarz, 20.03.2020 |
| Korekta: | Mgr inż. Maciej Stępień, programista, HSBC Service Delivery |